CRANFIELD UNIVERSITY

MICHAIL DIAKOSTEFANIS

INTERNET OPERATION OF AERO GAS TURBINES

SCHOOL OF ENGINEERING

PhD Thesis
Academic Year: 2013 – 2014

Supervisors: Dr T. Nikolaidis, Dr M. Stillwell, Dr S. Barnes,
Pr P. Pilidis
October 2014

CRANFIELD UNIVERSITY

SCHOOL OF ENGINEERING

PhD Thesis

Academic Year 2013 - 2014

MICHAIL DIAKOSTEFANIS

INTERNET OPERATION OF AERO GAS TURBINES

Supervisors: Dr T. Nikolaidis, Dr M. Stillwell, Dr S. Barnes,
Pr P. Pilidis

October 2014

ABSTRACT

Internet applications have been extended to various aspects of everyday life and offer services of high reliability and security. In the Academia, Internet applications offer useful tools for the remote creation of simulation models and real-time conduction of control experiments. The aim of this study was the design of a reliable, safe and secure software system for real time operation of a remote aero gas turbine, with the use of standard Internet technology at very low cost.

The gas turbine used in this application was an AMT Netherlands Olympus micro gas turbine. The project presented three prototypes: operation from an adjacent computer station, operation within the Local Area Netwok (LAN) of Cranfield University and finally, remotely through the Internet. The gas turbine is a safety critical component, thus the project was driven by risk assessment at all the stages of the software process, which adhered to the Spiral Model. Elements of safety critical systems design were applied, with risk assessment present in every round of the software process.

For the implementation, various software tools were used, with the majority to be open source API's. LabVIEW with compatible hardware from National Instruments was used to interface the gas turbine with an adjacent computer work station. The main interaction has been established between the computer and the ECU of the engine, with additional instrumentation installed, wherever required. The Internet user interface web page implements AJAX technology in order to facilitate asynchronous update of the individual fields that present the indications of the operating gas turbine.

The parameters of the gas turbine were acquired with high accuracy, with most attention given to the most critical indications, exhaust gas temperature (EGT) and rotational speed (RPM). These are provided to a designed real-time monitoring application, which automatically triggers actions when necessary.

The acceptance validation was accomplished with a formal validation method – Model Checking. The final web application was inspired by the RESTful architecture and allows the user to operate the remote gas turbine through a standard browser, without requiring any additional downloading or local data processing.

The web application was designed with provisions for generic applications. It can be configured to function with multiple different gas turbines and also integrated with external performance simulation or diagnostics Internet platforms. Also, an analytical proposal is presented, to integrate this application with the TURBOMATCH WebEngine web application, for gas turbine performance simulation, developed by Cranfield University.

Keywords:

Asynchronous, AJAX, Remote, Real-time, Spiral, Safety Critical, Risk Assessment, Software Process, Formal Validation, Model Checking

ACKNOWLEDGEMENTS

Theoretically this work was considered to a one researcher's project and at the very early stages I did have this feeling, as the subject was until then mostly unknown to me. However, as time and work went on, I realized that I was not working alone, but in collaboration with numerous other individuals, who contributed in different ways to the completion of this work. It is all these people then who I wish to thank and express my gratitude for their support.

First of all I wish to individually thank my three Supervisors, starting with Pr Pilidis, who honoured me by trusting me and assigning me this task, which was originally his own idea. He acted more than a Coordinator between the two involved Departments, as he was always willing to spare his limited time with me and support me with valuable advice and corrections.

Dr Nikolaidis, as the Power and Propulsion Supervisor, has vastly contributed to the evaluation of the gas turbine performance aspects that should be considered for the project.

Then it was Dr Barnes from the AMAC Group, who guided me closely during my first steps into software engineering and was always there to listen to my numerous questions, many of which later on I realized that they were trivial.

My special thanks to Dr Stillwell, who replaced Dr Barnes after the latter left Cranfield, but his knowledge in Internet applications development and his patience with me, played a major role for the accomplishment of this Thesis.

Dr G. P. Liu, from Glammorgan University. Although we met only once at an early stage of the project, he had the kindness to demonstrate the NCSLab application, developed under his supervision. His advice then was abstract but crucial for me.

Mr Charnley and all the Technicians in the test area for their immense support for the use of available equipment and facilities. Also, it would be an omission not to thank Mr Stan Collins, who had the kindness to give me some software modules he had designed, in order to reuse them in my application.

A special reference should be addressed to Sebastian Spratt, a very talented young man from Wootton Upper School. As a Nuffield Scholar, he designed the host web pages of the application, based on the guidelines provided by me.

I also want to thank my lovely daughters, Antonia and Evangelia, who both not only tolerated my absence from every day family life, but they were as well happy to help me with minor details of the Thesis layout.

Additionally, I have to mention the contribution of our family dog, Princess, who has been my patient audience with whom I discussed all my newly conceived ideas for the project, whilst we were having long relaxed walks.

Finally, there are no words to describe the contribution of my beloved wife Nektaria to the accomplishment of this Thesis. Even with all the support that I had from Cranfield and my Supervisors, it would never be possible for me to complete the work without the infinite support from her. She has been there for me throughout the whole period of my studies in Cranfield. She has shared hours with me helping me arrange this Thesis and took on all the family obligations, while I was absorbed with my studies.

I dedicate this work to my wonderful wife Nektaria and to the memory of my parents, Antonios and Kalliopi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

A	Abstractness
AC	Alternating Current
ACAS	Ambient Conditions Acquisition System
AJAX	Asynchronous JavaScript Xml
API	Application Programming Interface
CGI	Common Gateway Interface
CMM	Capability Maturity Model
CPU	Central Processing Unit
CSS	Cascade Styling Sheets
CTL	Computation Tree Logic
DAQ	Data Acquisition
DC	Direct Current
DDOS	Distributed Denial of Service
DLL	Dynamic Link Library
DMZ	Demilitarised Zone
DOS	Denial of Service
DTMC	Discrete Time Markov Chain
ECU	Electronic Control Unit
EDT	Engine Data Terminal
EGT	Exhaust Gas Temperature
EIS	Engine Interface Software
EISI	EIS Interface
EOCS	Engine Operation Control System
ETA	Event Tree Analysis
FAA	Federal Aviation Administration
FMEA	Failure Mode And Event Analysis
FMECA	Failure Mode And Criticality Analysis
FTA	Fault Tree Analysis
FTP	File Transfer Protocol
gc	Garbage Collection
GTH	General Test Harness
HAZOP	Hazard And Operability
HTML	Hyper Text Mark Up Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
I/O	Input / Output
IDS	Intrusion Detection System
I	Instability
ISASL	International Standard Atmosphere Sea Level
IT	Information Technology
Java SE	Java Standard Edition

JDK	Java Development Kit
JNA	Java Native Access
JNI	Java Native Interface
JSF	Java Server Pages
JSP	Java Server Faces
JVM	Java Virtual Machine
LAN	Local Area Network
LOC	Lines of Code
LOCM	Lack of Cohesion Of Methods
LTL	Linear Temporal Logic
M	Fundamental Matrix
MEOCS	Main Engine Operation Control Software
MSIS	Main Supply Interface System
NBD	Nested Block Depth
NCP	Number of Classes In Package
NI	National Instruments
NUSMV	New Symbolic Model Verifier
OS	Operating System
OPPD	On Line Power Plant Diagnostics
P	Transition Probability Matrix
PC	Personal Computer
PCN	Shaft Rotational Speed
pfd	Probability of Failure On Demand
PFMEA	Procedural of Failure Mode And Event Analysis
POJO	Plain Old Java Object
Q	Reduced Transition Probability Matrix
QOS	Quality of Service
R	Reliability
RESTful	Representational State
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RPM	Revolutions Per Minute
RTP	Real Time Protocol
SE	Standard Edition
SMV	Symbolic Model Verifier
SOAP	Simple Object Access Protocol
SRS	System Requirements Specification
SSH	Secure Shell
SSL	Secure Sockets Layer
t	Time
TBO	Time Between Overhaul
TCP	Transmission Control Protocol
T6	Exhaust Gas Temperature
TSS	Trouble Shooting System

UDP	User Datagram Protocol
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Unit Test Frame
V&V	Verification And Validation
VG	Cyclomatic Complexity
VISA	Virtual Instrument Software Architecture
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WMC	Weighted Methods Per Class
WSDL	Web Service Definition Language
XML	Extensive Mark Up Language
YUI	Yahoo Users Interface

GREEK SYMBOLS

α, β	Parameters of Beta Function
δ	Referred Pressure
θ	Failure Rate, Referred Temperature
μ	Average
σ	Standard Deviation

1 INTRODUCTION

This chapter describes the scope and the structure of the Thesis, along with a brief introduction to the gas turbine which was used in the project.

1.1 Scope

The Thesis presents the theoretical and experimental work accomplished in order to design a working software system that enables the flexible control of the operation of an uninstalled gas turbine remotely, with utilisation of Internet technology. When completed, the work produced 3 prototypes of the newly designed system: a local operating version from an adjacent computer station, a local area network (LAN) version and finally a web version. Initially, the problem is defined and appropriate literature review is presented. The experimental work is described along with the corresponding methodology in different self contained chapters, adapted to the applied software process model. Finally, the results and the final outcome are discussed, and potential recommendations for future work are suggested.

The web application was designed with provisions for generic applications. It can be configured to function with multiple different gas turbines and also integrated with external performance simulation or diagnostics Internet platforms. Also, an analytical proposal is presented, to integrate this application with the TURBOMATCH WebEngine web application, for gas turbine performance simulation, developed by Cranfield University. This combination would produce a very powerful remote experiment platform, where the theoretical modeling of an aero gas turbine will be compared and adapted to the performance of an actual engine in real time. Moreover, the combined applications could form a remote diagnostic and assessment platform for in-service aero gas turbines, reducing the necessity for physical presence of experienced and specialized personnel. In a more extended version, this application could be configured to operate UAV's through standard Internet technology.

1.2 Thesis structure

The Thesis comprises 12 chapters, divided into sub sections. The first 2 chapters (INTRODUCTION, AIM AND OBJECTIVES) are self explanatory.

Chapter 3, LITERATURE REVIEW, includes the literature review related to the application. In section 3.1, the problem is identified, after research and discussion of various similar projects and applications, based on information and publications available in the public domain. Section 3.2 is a very brief introduction to gas turbines, the potential risks in their operation and monitored parameters that are crucial for safety. Section 3.3 includes a research about risk analysis and methods used in industrial or aeronautical applications. Section 3.4 presents how risk analysis may be adapted to cover software related systems and what are the arguments about the effectiveness, according to the available literature. In section 3.5 the software process is discussed by addressing the key points of the procedure and how different approaches can be more or less appropriate for different types of applications. In section 3.6 a discussion about software tools suitable for interfacing real time mechanical systems takes place, stressing their advantages and disadvantages, always based on the literature available in the public domain. Section 3.7 briefly presents the Internet capabilities and the advantages or disadvantages of the various protocols. Finally, the effectiveness of the existing techniques for security over the Internet is discussed in section 3.8, along with methods of enhancing the performance.

CHAPTER 4 describes the software process and applied model, the architecture of the system and the requirements. Section 4.1 describes the methodology of the software process. Section 4.2 describes the methodology followed for the definition of the application requirements, the procedural risk assessment, the requirements elicitation and the derivation of the System Requirements Specification (SRS). The next section (Section 4.3) describes the architecture of the project, along with the modelling methodology applied. Section 4.4 includes the physical risk assessment of the project and the feasibility study. Section 4.5 presents the validation plan of the overall application and section 4.6 refers to the validation of the requirements. Finally, section 4.7 includes a discussion about results of the previous tasks.

CHAPTER 5 describes the part of the software process involved with the interfacing of the gas turbine with the PC. Section 5.1 describes the methodology followed and section 5.2 presents the risk assessment for this phase. The next section (Section 5.3) describes the analysis, the design and the implementation of the software, followed by Section 5.4, which describes the validation of the designed programs. Finally, section 5.5 includes a discussion about the results produced in this chapter.

CHAPTER 6 describes the part of the software process involved with the design of the next tier of software, the Engine Operation Control Software (EOCS). Section 6.1 describes the methodology followed and section 6.2 presents the risk assessment for this phase. The next section (Section 6.3) describes the analysis, the design and the implementation of the software, followed by the design of the General Test Harness (GTH). Section 6.4 describes the validation of the designed programs and section 6.5 includes a discussion about the results of the previous tasks.

CHAPTER 7 describes the part of the software process involved with the design of additional features of the system. Section 7.1 describes the methodology followed and section 7.2 presents the objectives and the risk assessment for this phase. The next section (Section 7.3) describes the analysis, the design and the implementation of the software, followed by Section 7.4, which describes the validation of the designed programs. Finally, section 7.5 includes a discussion about the results of the previous tasks accompanied by a presentation of the LAN version.

CHAPTER 8 describes the part of the software process involved with the design of the web application and the integration with the other designed modules. Section 8.1 describes the methodology for this task and section 8.2 presents the risk assessment. The next section (Section 8.3) describes the analysis, the design and the implementation of the software, followed by Section 8.4, which describes the validation of the designed programs. Finally, section 8.5 includes a discussion about the results of the previous tasks.

CHAPTER 9 presents the final part of the software process, which involves the acceptance validation of the final application. Section 9.1 presents the objectives and the risk assessment for this phase and section 9.2 describes the methodology followed. The next section (Section 9.3) describes the requirements testing and final testing. In section 9.4 a proposed approach for probabilistic analysis of the reliability of the system is presented and finally, section 9.5 includes a discussion about the results of the previous tasks.

CHAPTER 10 is a presentation of the development capabilities and suggested evolution of the system. Section 10.1 outlines how it will be capable of accommodating different gas turbines, section 10.2 describes the ability of configuration with multiple gas turbines and test stations simultaneously, and finally, section 10.3 explains how the program can be integrated with the TURBOMATCH Web Engine, an Internet gas turbine performance simulation tool developed by Cranfield University, for access through the Internet.

CHAPTER 11 presents the general results of the project, which are discussed herein. Section 11.1 discusses about the SRS and the general risk assessment, 11.2 about the implementation rounds, 11.3 about the overall validation procedures and 11.4 about the Installation of the application and the handling of the data between the hardware and software components. Finally, section 11.5 explains how the application may be optimised for different aspects.

CHAPTER 12 outlines the overall conclusions of the project and presents several recommendations for future improvement. Section 12.1 presents the conclusions and section 12.2 indicates the future work recommendations.

1.3 The AMT Netherlands Olympus gas turbine

The gas turbine used for the project is an Olympus micro gas turbine manufactured by AMT Netherlands (Figures 1, 2), used in radio controlled aircraft, remote heat/power generators or auxiliary power units [5].

Figure 1 - The Olympus AMT engine

Figure 2 - The Olympus AMT powering a Boeing 777 model

It is a miniature single spool turbojet gas turbine with an overall length of 270 mm, maximum diameter of 130 mm and maximum thrust of 190 N at maximum RPM (110,000). The single stage centrifugal compressor delivers a pressure ratio of 4:1 and handles a maximum value of mass flow 400 gr/sec [6]. It is controlled by a closed loop control Electronic Control Unit (ECU). The engine has a TBO cycle of 50 operating hours and already had accumulated 35 when the project commenced, posing a limiting factor to the amount of time available for running and testing.

2 AIM AND OBJECTIVES

The aim, objectives and contribution of this PhD study are outlined and explained in this chapter.

2.1 Aim

The aim of this project is to design a software system which will enable safe, secure and reliable remote operation of a gas turbine situated in a test facility, through the Internet, without tight coupling of the individual software components and with capability of running with browsers as thin clients - thus without the necessity to download any applets or other locally installed programs to run.

2.2 Objectives

The achievement of the final goal of the project requires the accomplishment of several objectives. These are:

- Hardware connection of the gas turbine with an adjacent PC control station - design of an appropriate interface to interact with it.
- Mitigation of the physical risks of operating the engine remotely
- Design of appropriate software components that will interact with the remote user and the engine. These components shall enable access through a local area network (LAN) and the Internet, and manipulate the input and output data accordingly to assure operation of the gas turbine within the desired operating limits.
- Plan and implement features of security against exposure and Internet threats.
- Includes provisions for a generic application to accommodate different type of gas turbines.

2.3 Contribution

This project introduces a novel application for real-time remote operation of aero gas turbines through the Internet, with standard browsers as thin clients and without the requirement to download additional software components. It demonstrates how the underlying physical risks, Internet security and network performance can be weighted and combined accordingly to provide a reliable tool. The above factors can be assessed and the project as a whole can be optimized for each one of them individually, thus the compromises posed may be examined, with respect to the level of the desired operability. The final application is a flexible design that may be integrated with

Internet gas turbine performance simulation applications, providing a powerful experimental and operation platform.

3 LITERATURE REVIEW

The literature review has focused on similar applications in the aviation sector, the industry and Academia, where many platforms for conducting remote experiments have also been designed. The review also investigates the various available options for Internet applications, the security against common Internet threats, software interaction protocols and the variety of techniques included in the software process. Very briefly the basic elements of gas turbine control theory and critical parameters are also discussed, along with software tools and programming languages suitable for real time systems.

3.1 Similar applications and problem definition

The information provided herein is based on material available in the public domain released by manufacturers and designers. For many applications, the available information is restricted, as it is considered to be confidential.

A large number of applications that enable remote access to engineering tools, based on Internet technology, are available in the market. Applications related to aero gas turbine operation monitoring and real time data collection from them may be found in the aviation field integrated with MRO packages, for monitoring and diagnostic purposes. In the power generation industry, there are systems that accomplish similar tasks in a different domain. In Academia, there are frameworks for developing distributed application that interface with hardware (mainly for control experiments). Also, there is a web-enabled gas turbine performance simulation platform, previously developed by Cranfield University. Many of the aforementioned applications rely on locally or centrally installed clients, others perform on thick clients, which require data to be downloaded and processed locally, or run by downloading and installing Java applets. Another category is Internet applications that run on regular Internet browsers acting as thin clients, but do not address the mitigation of high safety risks, like those underlying in the remote operation of a gas turbine.

There are situations where central control and monitoring of operations is required hence the centralised clients are very suitable. However, this configuration does not provide flexibility. When an application requires applets or other applications to be locally installed, the risks for the security of the user’s computer may increase. The safety of applets was recently questioned, when vulnerabilities were revealed, according to Oracle’s security alert CVE-2013-0422 [118]. According to the alert, vulnerabilities in security could allow for remote exploitation without authentication. Among other corrections, the Security Level was set from "Medium" to "High", which implies that the user is always prompted before any unsigned Java applet or Java Web Start application is run.

The current project introduces a light user friendly application, which enables the operation of a gas turbine through the Internet from any web browser, without the necessity of downloading and locally storing data or programs. It is very simple for the end user and on the other hand, it automatically caters for the safety of the operation with programmatic intervention when required.

3.1.1 Applications in the aviation sector

The development of contemporary MRO packages for airlines more and more focuses on the capability to access, retrieve and manipulate data at real time for monitoring or quality purposes. An example of this can be seen in manage – m Websuite developed by Lufthansa Technik, integrated with their MRO packages, which is a toolbox of web – based tools allowing for the management of the technical operations of an airline fleet. Operations as quality monitoring, reliability trends and status reports can be obtained via web services [90]. Manage – m can provide troubleshooting support by using data of the aircraft and its engines condition from on – board computers, even when the aircraft is airborne [90].

3.1.2 Applications in power generation

A similar application to the project described herein and with more details available in the public domain, is the Siemens SPPA – T3000 Control System for power plant management and control (Figure 3). As presented by Siemens, this system provides real time data at any place and any time, single software system imaging and simplification of the system architecture, rendering it easy to learn and use [130]. According to a research study from Karlsruhe University Germany, supervised by Siemens and available in the public domain, SPPA – 3000 is based on Java and XML and the application server can be accessed by any web browser. The web page of the application server consists of several Java applets. The system integrates different components of the plant around one main source named the Embedded Component Services [62]. The design follows a layered architecture with the following layers:

- Instrumentation: sensors and electronic devices
- I/O devices: digitalise analogue signals and transfer them upwards and vice versa
- Automation Servers: control devices and handle data to and from other Automation servers and upwards or downwards
- Application Servers (Web Servers): Windows 2003 OS, communicate data from the clients to the system and vice versa

- Thin clients: web browsers with Java applications

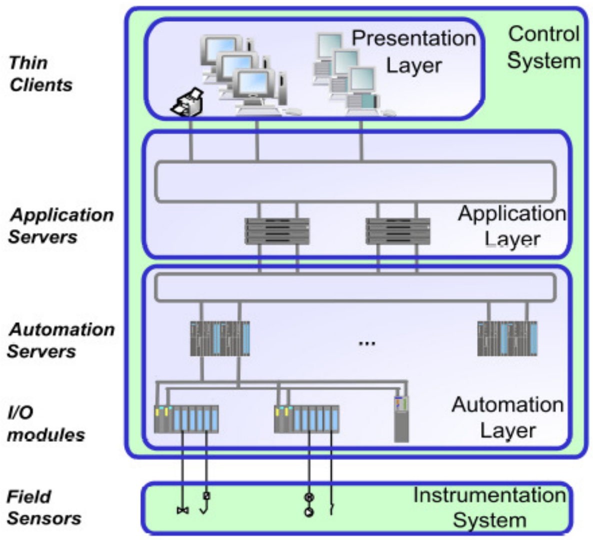


Figure 3 – SPPA-T3000 Architecture [62]

As presented by the resource from Karlsruhe University, the key points of safety and security of the system are the following [62]:

- Physical safety
- Communication between servers and clients only through HTTPS with SSL
- Access via the Internet through VPN tunnel and firewall
- Reference for introduction of SSH (Secure Shell) in the future. Applicable to client – server model
- Intrusion Detection System (IDS) as proposed

It is noted at this point that the aim of the Karlsruhe University research study [62] was the introduction of IDS and the final proposal included the selection of Snort, which is a signature based detection system. Additionally, the system implements a redundancy philosophy for communication between layers based on a set on X400 switches, which act as a redundancy manager system in a formed ring topology between the highways of connection [62] (Figure 4).

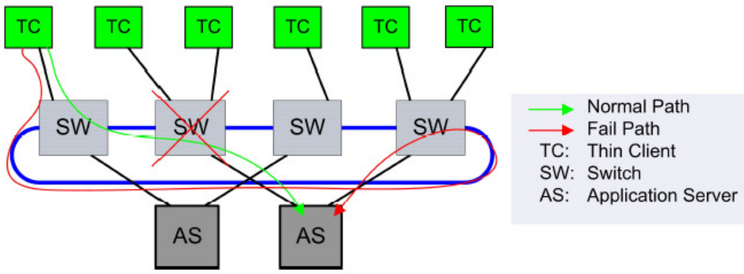


Figure 4 – Failsafe ring topology [62]

ALSTOM has developed an integrated support system for power plants, the Plant Support Center [3]. It provides direct access to a global network of technical specialists to support troubleshooting, equipment trends analysis or to provide support during assessment [1]. The system provides 24/7 operation support, monitoring and diagnostics, and power train vibration remote assessments. Data from site are transmitted to the remote diagnostics central server via a secure connection at least once every 24 hours. Any deviation observed after the assessment of the data by the diagnostics tools at the Support Center, appropriate engineers or the plant operators are notified for action [2] (Figure 5).

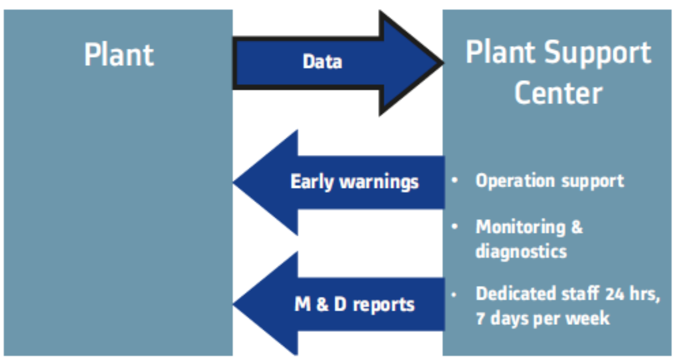


Figure 5 – ALSTOM Plant Support Center operation [2]

A closely related application to the PCS is OPDD (On Line Power Plant Diagnostics), a development project at Alstom Power Finland Oy, designed for gas turbine combined cycle power plants, but can also be adapted to other types as well [81]. The data are transmitted via a secure connection to the support centre, where they are assessed against data from stored models (Figure 6). The information is displayed on appropriate clients [81] (Figure 7).

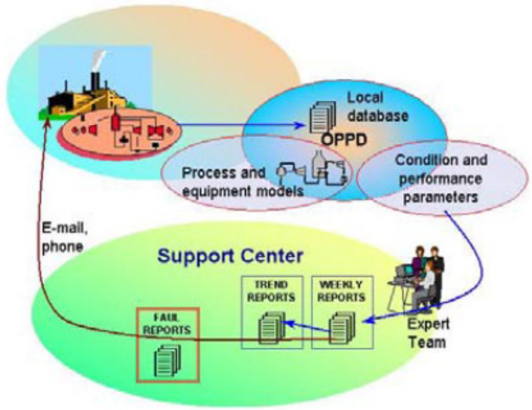


Figure 6 – OPPD for ALSTOM Finland [81]

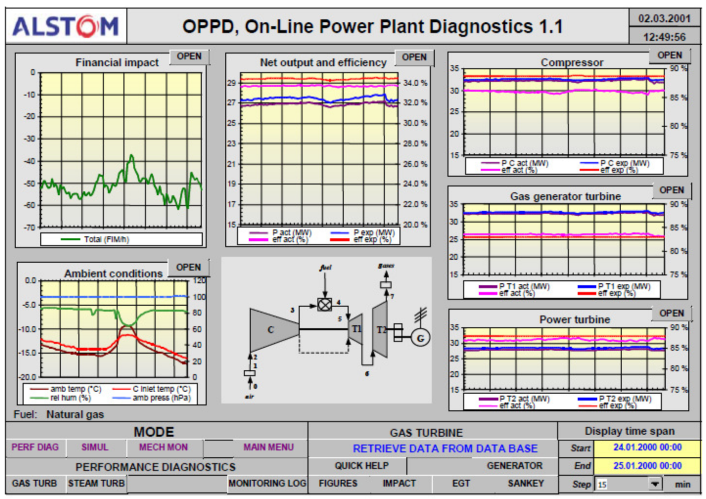


Figure 7 – OPPD for ALSTOM Finland – Diagnostics display [81]

3.1.3 Academic applications

Another similar application is the NCSLab, developed by Glammorgan University (Figure 8). It is a control engineering experiment platform which consists of 6-tier architecture, based on MATLAB – Simulink. The experiment simulation takes place on MATLAB servers wrapped in Java RMI's in order to communicate with the central server. It also includes a scheduled user access methodology inspired by the LINUX resource read/write privilege system. However, only one person can have full control privilege at a time. The redundancy of servers and device status detector module ensures network stability [124]. The layers of the system are shown below:

- Controlled Devices

- Control Units
- SOCKS proxy server to override firewall, if devices are installed in LAN
- Experiment servers, which execute the control demands from the main server and transfer real time data to the web MATLAB servers. These provide remote simulation and compilation for the main server, avoiding MATLAB installation on local PC's
- Main server, which accepts requests from clients and displays generated dynamic web pages
- Clients (web browsers that employ AJAX and Flash technologies)

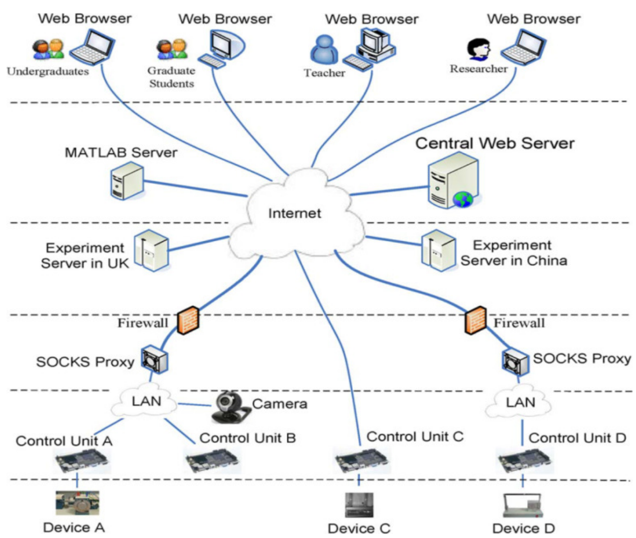


Figure 8 – NCSLab architecture [124]

It is worth to be noted at this point that according to Qiao et. al. (2010) [124], SOAP is not suitable for displaying real – time data to the client. Hence AJAX technology was implemented in order to achieve real – time display with automatic renewal of the page at every change. The transport layer protocol used for real – time data transfer was UDP whilst for non – real time, TCP.

Various other approaches have been designed to accommodate the capability of remote lab operation, mainly for control experiments. Ko et. al. (2001) [84] have developed a web – based laboratory for control experiments on a coupled tank apparatus (Figure 9). This application was based on LabVIEW and TCP for communication between client and server, instead of Common Gateway Interface (CGI). TCP remains in contact until the client's request for discontinuing, whilst CGI – which is a per session basis protocol – would close after the response of the server. The data transfer between client and server is achieved with the employment of Java

Applets on the client end, which activates the TCP connections at the appropriate LabVIEW functions at the server end.

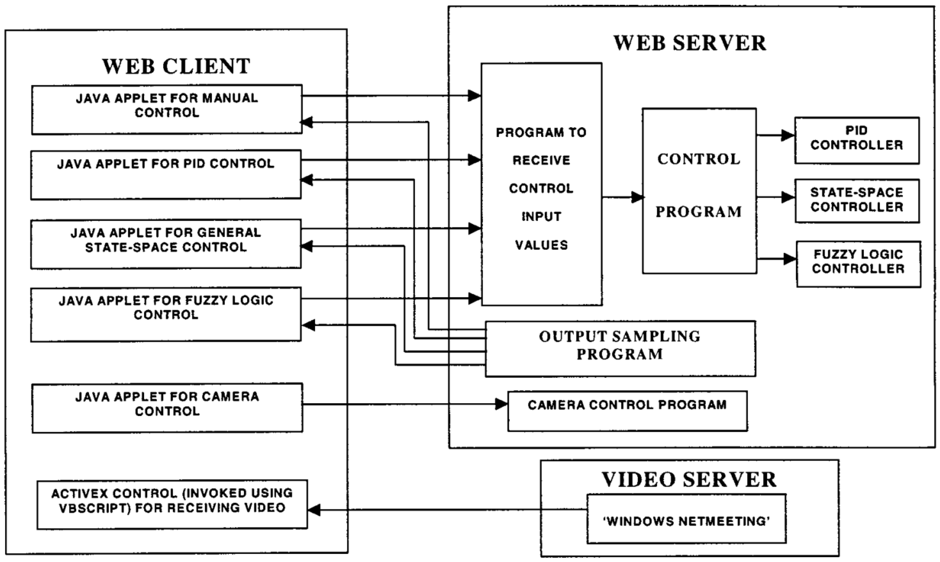


Figure 9 – Web based laboratory for control of coupled tank apparatus [84]

Similarly, Okajima et. al. (2006) rely on Java Applets on the client side for their WebLab Development Using a /DE9LHZ and -DYD integrated solution for Kyatera Network [108]. This design also incorporates LabVIEW for the conduction of the control logic and communication between Java and LabVIEW is achieved with TCP/IP.

Another platform for remote control experiments has been introduced by Elmerabete et. al (2010) (Figure 10). Real time control is obtained with LabVIEW and TCP/IP communication between the client and the server [34].

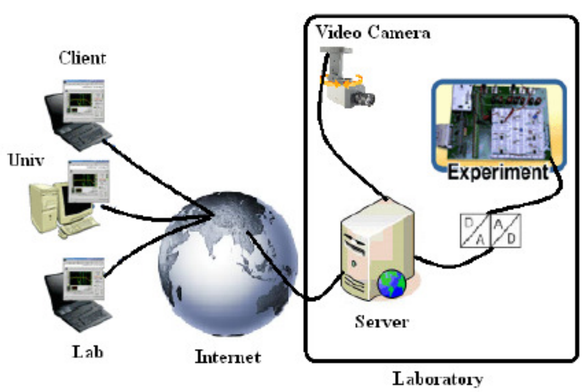


Figure 10 - Application of Real-time Control System Using LABVIEW in Distance-Learning [34]

3.1.4 The Turbomatch WebEngine

The TURBOMATCH WebEngine is an Internet application developed by Cranfield University, for the creation of gas turbine performance simulation models [7]. It utilises TURBOMATCH in the server side, a powerful gas turbine performance simulation tool developed by Cranfield University as well. The WebEngine offers ease of use with no local installations required (Figure 11).

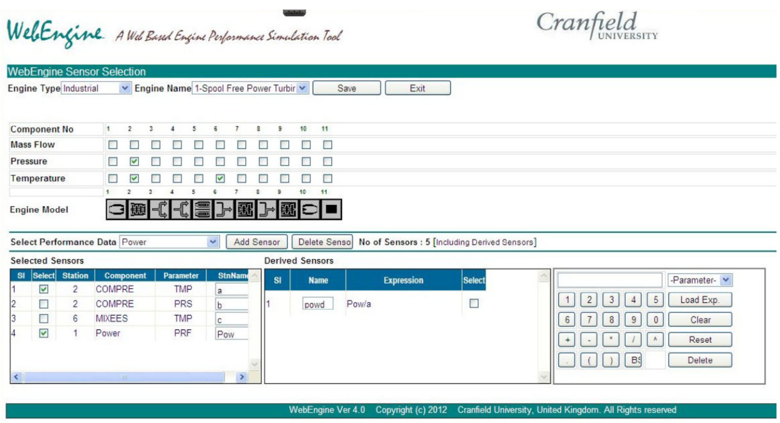


Figure 11 – The User Interface of the TURBOMATCH WebEngine [7]

It is based on client – server architecture, with TURBOMATCH as the core solver behind the GUI. As stated by Apostolidis (2013) [7], it has 2 major advantages. One is the modular architecture, which enables interchangeable engine components and future development. The other is the ability to integrate with other tools, such as optimisers and flow solvers. TURBOMATCH is built in a DLL file that is invoked by the Virtual Engine Console (Figure 12).

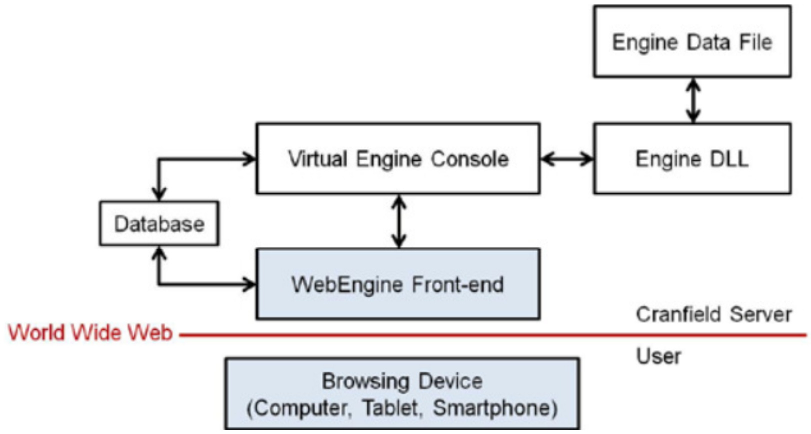


Figure 12 - The WebEngine Architecture [7]

The user uses the drag and drop features to easily build the gas turbine model, which comprises of several individual bricks representing the major components. The WebEngine includes capability for design point and off – design performance, with the user entering the required parameters for each brick (Figure 13 and Figure 14).

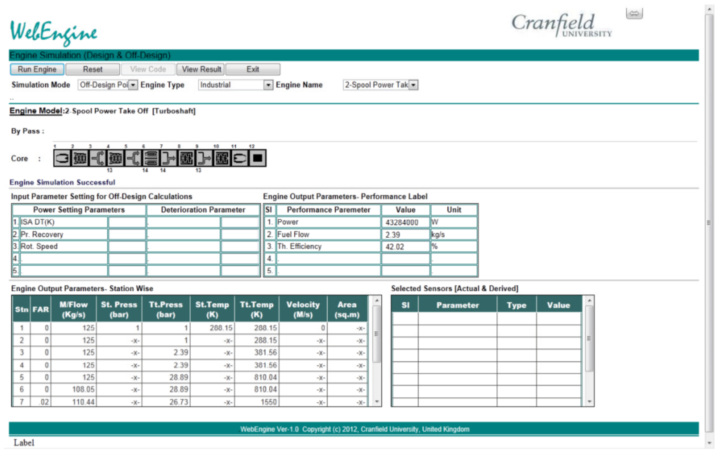


Figure 13 – GUI of the WebEngine (Off design performance simulation) [7]

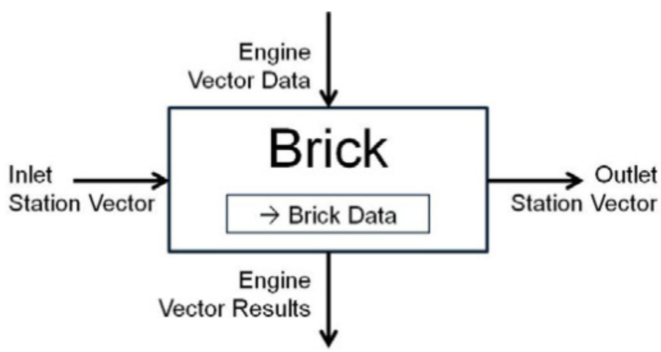


Figure 14 – TURBOMATCH brick and related data [7]

3.2 Gas turbines and basic performance monitoring elements

3.2.1 Basic principles

The various types of gas turbines are mainly distinguished in civil aviation, military aviation, industrial or marine. The major aim of a gas turbine is either the production of thrust (turbojet or turbofan aviation gas turbines) or power (turboprop, industrial, marine). There are several criteria for performance evaluation of a gas turbine, such as the propulsive efficiency, which is the ratio of the useful propulsive energy or thrust power against the kinetic energy of the jet and the specific thrust, which depicts the thrust per unit mass of air absorbed by the engine [129]. These parameters are used to describe the efficiency of the thermodynamic cycle of a gas turbine, known as the Brayton cycle [120].

Variables which influence the efficiency of the cycle are mainly the inlet air mass flow, the overall pressure ratio of the compressor, the heat input which relates to the fuel quantity and the expansion ratio at the turbines. The internal arrangement of the gas turbine determines the amount of compression work required, which is consumed from the total work produced by the engine [120]. According to Walsh et. al. (2004) [145], all the variables which describe the can be distinguished in groups of non - dimensional, semi – dimensional, referred, scaling or combined. An analytical table of these parameters can be found in Walsh et. al. chart 4.1 [145].

3.2.2 Steady state and transient performance

The state of an operating gas turbine can be distinguished in steady state and transient performance. Steady state refers to the condition of operation where the parameters of the engine are constant relative to time. On the contrary, transient performance refers to the operating regime where the engine parameters alter with time [145]. Practically, it is the time intervals when throttle settings have been changed thus altering the parameters in order to obtain the newly selected steady state condition. As Walsh et. al. (2004) describe for a single spool turbojet gas turbine operating steady state, when the control system suddenly increases fuel flow, then the turbine power output increases because of the increased temperature. This exceeds the power output required to drive the compressor, auxiliaries and mechanical losses, rendering a case of unbalanced power [145]. The engine performance becomes a function of two independent parameters: fuel flow and rotational speed [120], hence requires a more complicated approach for parameters calculation, compared to steady state performance.

3.2.3 Failure modes

It is noticeable that compromise between independent parameters of the gas turbine operation is necessary and imposed by operating limits of critical rotating parts. Although increasing rotational speed enhances performance, it increases centrifugal stresses on rotating blades. These are steady state stresses which are combined with the dynamic stresses due to aerodynamic loads and vibrations and promote the fatigue failure mode (either LCF or HCF). Also as turbine entry temperature increases, the performance improves with the penalty of exceeding the operational limits of the turbine blades and vanes materials, promoting the failure mode of creep [129]. A general view of the failure modes of a gas turbine is shown in Figure 15.

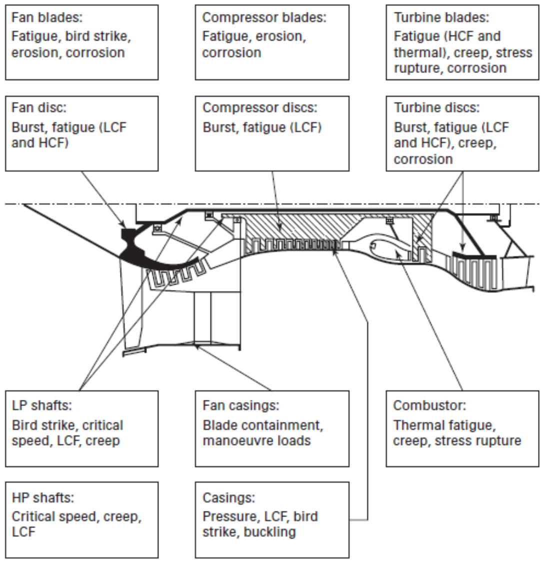


Figure 15 – Gas turbine failure modes [129]

As gas turbines evolved through the decades, the necessity of sophisticated electronic closed loop control systems appeared. Modern gas turbine control systems consist of four control components, which are the controller, sensors, actuators and accessories. They aim to control the thrust at the desired value (Figure 16). However, as thrust cannot be practically measured in flight, control is obtained by varying fuel to regulate either the rotational speed or the pressure ratio of the engine, which are indicators of the actual thrust magnitude [78].

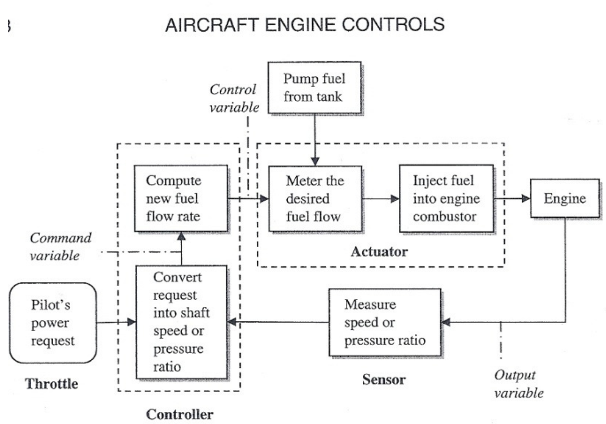


Figure 16 – Functional diagram of a simplified engine control system [78]

3.2.4 Gas turbine control systems

Control systems apply alarm rules in order to detect values above certain limits during the engine operation. There are four unusual patterns of data points:

- The instability pattern, where points are present outside the control limits,
- The mixture pattern, characterised by the absence of points near the centreline,
- The stratification pattern, with absence of points near the control limits and
- The trend pattern, where an upward or downward trend is present.

The most noticeable for the present study is the instability pattern, which means the existence of data points outside the control limits. The rules which are most widely applied for the classification are the Western Electric rules (Figure 17). They divide the control band into three zones where zone A is the area enclosed within 2 and 3 standard deviations (σ) of the parameter, zone B, which stays between 1σ and 2σ and C, between ±1σ. A guideline for the alarm generation is: any point beyond zone A, 2 out of 3 consecutive points in zone A, 3 out of 5 consecutive points in zone B and 8 consecutive points on the same side of the centreline [78]. Typical input and output parameters of a large bypass ratio turbofan gas turbine are shown in Table 1.

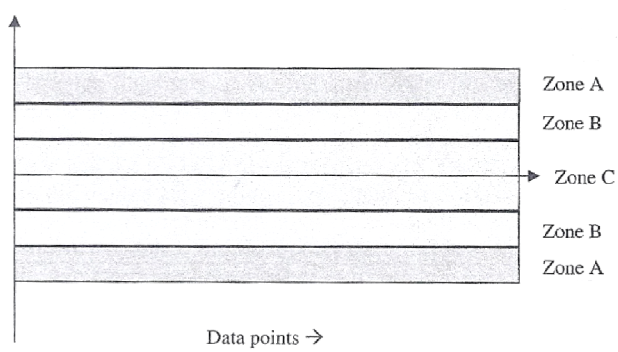


Figure 17 – Western electric rules [78]

Finally, the monitoring of the gas turbine is essential for detection of abnormal conditions, such as fault, anomaly or failure [78]. These can be detected by observing deviation of certain engine parameters beyond the predetermined threshold of tolerance. Hence parameters such as pressure ratio, pressure and temperature at various stages of the engine, fuel flow and vibration levels are monitored and acquired in order to perform diagnostics and prognostics (Table 1). The former refers to discovering the current and past abnormalities whilst the latter refers to estimation of future performance [78]. Examples of diagnostic methods are linear or non linear Gas Path Analysis, statistical inference based on the Maximum Likelihood estimate and artificial neural networks [88]. Various systems based on engine parameters trend analysis have been developed for implementation of prognostics programs.

Table 1 –Indications and required input of typical Turbofan Engine CFM56-CA3 [32]

Engine Input	Engine Output via FADEC
Throttle angle	**Measurements indicated to user**
Starter	LP Rotational Speed
Electrical Power supply switch	HP Rotational Speed
Fuel supply switch	LPT STG. 2 Total Temperature
Ambient Static Pressure	Engine Oil Temperature
Inlet Static Pressure	Engine Oil Pressure
Inlet Total Temperature	Fuel Mass Flow delivered to the engine
	Condition indicated to the user
	Vibration Level of the HP shaft
	Vibration Level of the LP shaft
	Fault report
	Measurements required for control or diagnostics
	HPC Inlet Total Temperature
	HPC Delivery Static Pressure
	HPC Delivery Total Temperature
	HPT Shroud Support Temperature
	Fan Outlet Static Pressure
	HPC Inlet Total Pressure
	LPT Outlet Total Temperature

3.3 Hazard Identification, risk analysis and risk mitigation

An important element of systems that are considered safety critical is risk analysis. The basic method of risk analysis is the identification and quantification of scenarios which may lead to hazardous conditions, and also the identification and quantification of the probabilities of these scenarios to occur, along with their consequences [10]. Hazard can be defined as the presence of any process abnormality which represents either a dangerous or a potentially dangerous state [68]. Risk could also refer to the statistical annual frequency for a particular hazard state [68].

3.3.1 Risk evaluation and mitigation

The approaches and individual methodologies for risk evaluation and hazard mitigation described herein follow guidelines applicable to power plant systems, aerospace, chemical and nuclear industries, as these are considered to be adequate for covering the risk evaluation and mitigation of the present project. The underlying risk in this project is increased, because it involves the risks relative to the operation of a gas turbine in a stationary test facility combined with the risks due to remote operation through the Internet.

A method often found in power plant, aerospace, chemical and nuclear systems is the Fault Tree Analysis (FTA) [68] [47]. This method is based on the identification of a top undesirable event that corresponds to a particular failure mode and consists of the individual events that contribute to the top event (Figure 18). FTA is a qualitative method in its nature but it can also provide a quantitative aspect as well [134]. According to BS IEC 61025, ‘FTA is particularly suited to the analysis of complex systems comprising several functionally related or dependent subsystems with different performance objectives. This is especially true whenever the system design requires the collaboration of many specialized technical design groups. Examples of systems to which fault tree analysis is commonly applied include nuclear power generating stations, aeroplanes, communication systems, chemical and other industrial processes’. The standard recommends the development of the fault tree early in the system design stage [20]. The logical analysis of the FTA is obtained with the implementation of Boolean reduction and the method of minimal cut sets [20] [68].

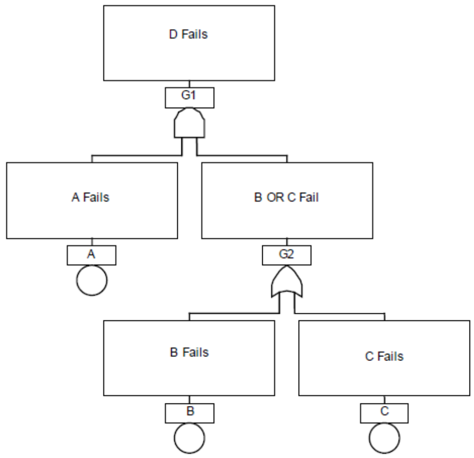


Figure 18 – A simplified FTA [134]

The Event Tree Analysis is related with FTA. Event trees are logical networks which do not include logical gates as the FTA and commence with the hazard state event (FTA top event) to assess the consequences to follow [68].

Hazard and Operability (HAZOP) analysis is a method mainly used by the chemical industry [47] and it deals with the identification of deviation from the design intent, also investigating the causes and assessing their consequences [20] (Figure 19). It is suggested that a HAZOP analysis is better to be conducted by a team.

STUDY TITLE: PROCESS EXAMPLE								**SHEET: 1 of 4**	
Drawing No.:			**REV. No.:**					**DATE:** December 17, 1998	
TEAM COMPOSITION:			LB, DH, EK, NE, MG, JK					**MEETING DATE:** December 15, 1998	
PART CONSIDERED:			**Transfer line from supply tank A to reactor**						
DESIGN INTENT:			**Material:** A **Source:** Tank for A	**Activity:** **Destination:**	Transfer continuously at a rate greater than B Reactor				
No.	**Guide word**	**Element**	**Deviation**	**Possible causes**	**Consequences**	**Safeguards**	**Comments**	**Actions required**	**Action allocated to**
1	NO	Material A	No Material A	Supply Tank A is empty	No flow of A into reactor Explosion	None shown	Situation not acceptable	Consider installation on tank A of a low-level alarm plus a low/low-level trip to stop pump B	MG
2	NO	Transfer A (at a rate >B)	No transfer of A takes place	Pump A stopped, line blocked	Explosion	None shown	Situation not acceptable	Measurement of flow rate for material A plus a low flow alarm and a low flow which trips pump B	JK
3	MORE	Material A	More material A: supply tank over full	Filling of tank from tanker when insufficient capacity exists	Tank will overflow into bounded area	None shown	Remark: This would have been identified during examination of the tank	Consider high-level alarm if not previously identified	EK

Figure 19 – HAZOP example [20]

Failure Modes and Effect Analysis (FMEA) is a method for the analysis of a system in order to identify the potential failure modes, their causes and any effects on the system performance [19]. According to the related standard [19], the term system may represent hardware, software (with their interaction) or a process. FMEA may be initiated when the system is defined enough to be presented as a functional block diagram. According to this, FMEA is extremely efficient when it is applied to the analysis of elements that cause a failure of the entire system or of a major function of the system. It can be used alone or as a systematic inductive method of analysis, to complement other approaches, especially deductive ones, such as FTA. It is met in nuclear, chemical [47] and other applications. As stated by Fullwood (1999) [47], FMEA does not account for human operator errors, but still, these are depicted in the analysis by the respective equipment failure mode. When including criticality analysis in FMEA, it is then considered as Failure Modes and Effect Criticality Analysis (FMECA) [47] [19]. Criticality is a function of severity and frequency of a failure mode [19] and it provides a tool for quantification of risk (Figure 20). HAZOP is a system-centred approach whilst FMEA is component centred. FMEA starts with a possible component failure and then proceeds to investigate the consequences of this failure on the system as a whole [19].

Frequency of occurrence of failure effect	Severity levels			
	1 Insignificant	2 Marginal	3 Critical	4 Catastrophic
5: Frequent	Undesirable	Intolerable	Intolerable	Intolerable
4: Probable	Tolerable	Undesirable	Intolerable	Intolerable
3: Occasional	Tolerable	Undesirable	Undesirable	Intolerable
2: Remote	Negligible	Tolerable	Undesirable	Undesirable
1: Improbable	Negligible	Negligible	Tolerable	Tolerable

Figure 20 – Risk – Criticality matrix [19]

3.3.2 Probabilistic risk analysis

Probabilistic risk analysis (PRA) is a tool for quantitative risk analysis applied in sectors such as aerospace industry, chemical processing, energy production and transportation. It provides a set of tools to deal with uncertainty and apply probabilistic methods to estimate frequency or probabilities of undesirable events for the purpose of risk evaluation [10]. Statistical inference holds an important role in PRA, as it is called to utilize a limited sample of experimental observations and make an inference about the appropriate probability distribution to describe it [10]. A correct probability distribution is crucial as upon it, the developed statistical data for risk quantification will rely.

3.3.3 Project risk analysis

Not only physical risk analysis but also project risk analysis has been rapidly growing during the past decades. Bedford (2001) [10] describes the necessity of such an analysis in the construction industry, where large scale infrastructure projects suffer from various uncertainties (political and financial). He states that application of this analysis can provide the management with quantitative insight into the different sources of uncertainty of the project, allowing for proper action. A typical example of a project risk management approach is the RISMAN method [10], which includes the following steps:

- Identification of uncertainties
- Quantification of uncertainties and countermeasures
- Calculation of project risk
- Calculation of the effect of countermeasures
- Decision making and risk handling

It can be observed that the above steps are more or less included in the HAZOP and FMEA methods. It is stated in the BS EN 60812 that FMEA can be used as a tool to study various processes (medical, laboratory, manufacturing, development, educational, etc.). In that case it is assumed as the Process FMEA or PFMEA. When PFMEA is performed, it is always done with regards to the process end goal or the target of a process. Each step within that process is then considered as a potential to produce an undesirable outcome of the other steps in the process or of the process end goal. Similarly, HAZOP has evolved to a tool which can be applied to assess wider applications like systems involving the movement of people by transport modes such as road and rail, examining different operating sequences and procedures, and assessment of administrative procedures in various industries [20].

3.4 Risk assessment and quality determination of software products

3.4.1 Software quality metrics

Quality determination of software products cannot be considered to be as straight forward as in the manufacturing industry. The determination of software quality has improved noticeably during the past 15 years. This improvement can be attributed to the development of techniques and standards that aim to specify and quantify quality for software products. Sommerville (2006) [122] indicates that parallelism of software and manufacturing quality cannot be obtained. That is because as software development is a complex concept with the product specifications difficult to be concretely defined in the initiation of the software process and software quality characteristics are not easily defined unambiguously [122]. According to the ISO 9126 standard, six key characteristics have been defined to describe quality: functionality, reliability, usability, efficiency, maintainability and portability [8]. The software quality metrics can be classified in two groups, product and process metrics. As Sommerville

(2006) describes, there is a complicated and close relation between process and product quality, where the quality of the latter depends on the quality of the former. Several organizations are involved in the development of software quality determination standards, some of them being US DoD, NATO, ANSI, BSI and IEEE [132].

Oliveira et. al. [109] (2008) identify the correlation of quality metrics for traditional software products with the so important physical metrics for embedded systems (performance, memory, energy, power, size, and weight). They outline several of the most important software quality metrics. The most notable of these are:

- Coupling
- Cohesion
- Lack of Cohesion of Methods
- Extendibility and reuse
- Population metrics
- Complexity

3.4.2 Safety critical systems

A safety - critical system is a system which, if it fails, it may cause life loss, injury or environmental impact. When designing a safety – critical system there are four major key points that characterize its dependability, along with all the quality features previously listed: availability, reliability, security and safety [132]. Availability expresses the ability of the system to run successfully at any given time whilst reliability can be defined as the probability it will execute as expected. It can be stated that security and safety are interrelated with reliability, as insufficient degree of the two latter can directly affect the former [132]. A generic approach for a safety life cycle for electrical, electronic or programmable electronic safety related systems is standardized by BS EN 61508 [18]. According to this standard, various phases of a safety life cycle include from initial concept through design, implementation, operation and maintenance to decommissioning. It introduces the concept of safety integrity levels which depict the target of the safety level activities to be implemented in the related functions and adopts a risk based approach to determine the safety integrity levels [18].

3.4.3 Software reliability prediction

Gokhale and Trivedi (2002) consider that computer industry has seen uneven progress, as ‘With the steadily growing power and reliability of the hardware, software reliability has been identified as a major stumbling block in the realization of highly dependable computer systems. When lives and fortunes depend on software, assurance of its quality becomes an issue of critical concern’ [54].

Several qualitative and quantitative methods have been applied for software reliability determination. For qualitative assessment, FMECA has been suggested, although not accepted by the whole software development community, as software components do not fail like mechanical ones [10]. Formal design and analysis methods are another approach which set up a discipline framework for specification programming, in order to reduce the chances of errors creation. Sneak analysis is described as a method or set of methods to trace design errors. It is regarded successful in finding problems that cannot be found by testing. Software testing is a crucial part of the development phase, inexplicably linked with the verification and validation process. It will be thoroughly discussed in Section 3.5.4. Error reporting is an important method accomplished by the users of the software via various systems [132].

The implementation of statistical tools for quantitative prediction of reliability has been controversial as software cannot follow the same engineering failure modes as physical or mechanical components do [10]. In mechanical engineering the various failure modes have been thoroughly studied and determined, whereas in software engineering, failure modes study and prediction of reliability are more recent fields. However, numerous estimation techniques have been developed, applicable for different cases. The goal of these estimators is to minimize the error between the actual value of the parameter and the estimated value [24]. According to Cangussu et. al. (2011) [24] these techniques can be mathematically classified in Bayesian Belief Networks (BBNs), Bayesian Parametric Approaches (BPAs), non-parametric Models and Classical Parametric Approaches [24]. The study of Cangussu et. al. (2011) [24] presents a ranking of different representative techniques from the 4 mathematical categories, based on subjectivity, cost, applicability, latency and expressiveness. The ranking is supported by an example using 4 scenarios based on 4 organisations of different maturity level, according to CMM [131]. For example, for scenario 1, which refers to maturity level 1, there are little data available and lack of expert knowledge. The importance of low cost and high applicability favours the Classical Parametric approaches and *CPA* AP4, thus the ED3M model is ranked in first position [24].

Meedeniya et.al. (2012) introduced a framework for evaluation of probabilistic models with uncertain parameters [94]. This framework applies Monte Carlo simulations to evaluate the reliability of a system based on its architecture specification. It has been expanded to utilise probabilistic model checking rather than using a dedicated reliability evaluation model with a specific evaluation technique. The main formalisms used in this approach are probabilistic temporal logics and Markov Models [94].

This work was a continuation of that of Meedeniya et. al. (2011), where a number of parameters (environmental factors or system usage) uncertainties in architecture-based quality evaluation were investigated [93]. The probabilistic model was constructed as a well established DTMC (discrete markov chain) and Monte Carlo simulation was then applied. The expected number of visits to the nodes of the DTMC quantifies the expectation of use of this component, during a single execution of the system. To estimate the overall reliability, Meeddeniya et. al (2011) followed the approach also presented by Gokhale et. al (2002) [54], where the expected reliability of an application of n components, while ignoring the second order architectural effects, is given by the following equation:

$$\boldsymbol{E}[\boldsymbol{R}] \approx, \prod_{i=1}^{n-1} \boldsymbol{R}_i^{m_{1,i}} \boldsymbol{R}_n$$

Equation 1 – Expected Reliability of an application with n components

When individual software components reliability is considered, various models are used to determine it. Goseva et. al (2001) [60] state that there is a class of models that estimate reliability from explicit considerations of non – failed executions, usually in the testing during the validation phase, where the software is not undergoing any changes [89], [96], [121]. In safety critical applications, the required reliability levels (10^{-9} /h or 10^{-5}/h) is unreachable hence it is generally agreed for a practical limit of a failure rate in the range of 10^{-2} /h or 10^{-4}/h to be aimed for in the assessment prior to operational use [60].

3.4.4 Avionics development

Software developers in avionics follow the RTCA/DO-178 document guidelines (Software Considerations in Airborne Systems and Equipment Certification) in order to obtain FAA approval [65]. DO-178C has now been released and addresses errors and inconsistencies present in previous editions [126]. The DO-178 document includes guidelines for the software lifecycle, the planning and development process, verification, configuration management and quality assurance. An important topic within the document is the definition of criticality levels of software, determined according to the extent of the damage caused in the case of a failure, ranging from E (no effect) to A (catastrophic) [69] (Figure 21).

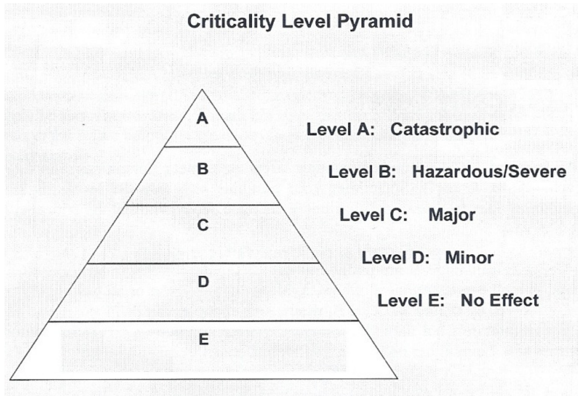


Figure 21 – Criticality Levels according to DO-178B [69]

Finally, the prediction of the inherent reliability of a variety of military electronic equipment and systems is based on MIL – HDBK – 217F [140]. This document establishes and contains uniform methods for this purpose and also provides a basis for comparison and evaluation predictions or competitive designs. It introduces two methods of reliability prediction, the Parts Stress Analysis and the Parts Count. The former requires a larger amount of detailed information and it is applicable in the later stage of design, whereas the latter requires less information and is suitable for early design stage [140].

3.5 Software development

The development of a software system is described by the software process, which is a group of activities leading to the production of a software product [132]. According to Sommerville (2006) [132], the key points of the software process are the software specification, the software design and development, the software validation and the software evolution.

3.5.1 Software process models

Generic models describe the software process, based on the form of the requirements and the mode of approach. A model that mostly resembles the classical engineering models is the waterfall model, where each of the activities of the process represents an individual phase. It requires concrete software specification, thus commitments need to be made at early stages of the process, rendering the approach inflexible to alterations and changes that occur during later stages [132]. With evolutionary approach, the specification, design and implementation, and validation phase interleave and an initially developed system is evolved to meet the requirements of the customer [132]. It does not require concrete requirements specification, hence the requirements may be revisited at any phase and be easily modified [8]. A model which utilizes reusable software components to build a new system is known as the component – based software engineering model. This approach integrates existing components instead of designing them from the beginning and it is very useful in the case of existence of such components relative to the product under design [132].

As changes of the requirements of large projects are inevitable, an iterative approach is more efficient to cope with the particularity of the software development. An example of an iterative software process is the spiral approach, introduced by Boehm (1988) [13]. Each loop in the spiral represents a particular phase of the whole process and for each loop one may distinguish four sectors. These are the setting of the objectives, the assessment and reduction of risk, development and validation, and planning for the next cycle (Figure 22). The explicit recognition of risk distinguishes the spiral from the other models. The arrangement of this model allows particular phases of the process to be revisited if required [132].

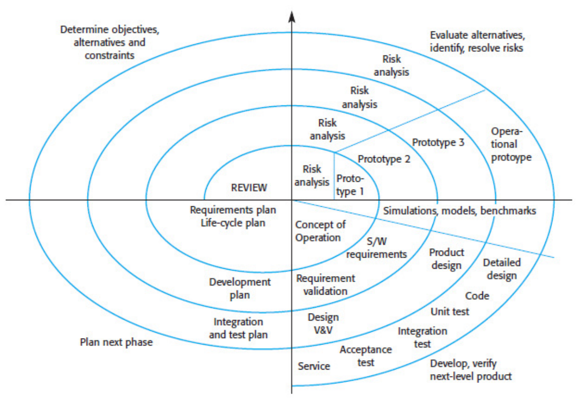


Figure 22 – The Spiral Model [132]

An application of the Spiral Model can be seen in the definition and development of the TRW Software Productivity System (TRW-SPS), an integrated software engineering environment of the American Military. The first cycle of the spiral refers to the feasibility study, the second cycle to the concept of operations and the third round includes the top level requirements specification [26]. The elements of round 1 of the spiral are shown in Figure 23.

Objectives	Double software productivity in five years
Constraints	$10,000 per person investment Within context of TRW culture • Government contracts, high tech, people oriented, security Preference for TRW products
Alternatives	Office: Private/modular/… Communication: LAN/star/concentrators/… Terminals: Private/shared; smart/dumb Tools: SREM/PSL-PSA/…; PDL/SADT/… CPU: IBM/DEC/CDC/…
Risks	May miss high-leverage options TRW LAN price/performance Workstation cost
Risk resolution	Extensive external surveys, visits TRW LAN benchmarking Workstation price projections
Risk resolution results	Operations concept: Private offices, TRW LAN, personal terminals, VAX Begin with primarily dumb terminals; experiment with smart workstations Defer operating system, tools selection
Plan for next phase	Partition effort into software development environment (SDE), facilities, management Develop first-cut, prototype SDE • Design-to-cost: 15-person team for one year Plan for external usage
Commitment	Develop prototype SDE Commit an upcoming project to use SDE Commit the SDE to support the project Form representative steering group

Figure 23 – Round 1 of the spiral process for the TWR – SPS project [26]

Another example of the spiral model may be seen in a project described by the Swedish University of Linkoping in one of their handbooks. It is suggested for a software project of the PUM course and the spiral includes 3 rounds [67]:

- First round: Pre – study and definition
- Second round: Architecture and design
- Third round: Implementation and test

3.5.2 System Requirements

The significance of the system requirements is very high as their ambiguity and preciseness defines the clarity of the project. Sommerville (2006) suggests a classification in User's Requirements, which are high level abstract requirements and System Requirements, which are a detailed description of the system functionalities. The system requirements can represent a whole phase of the spiral model and the engineering process to produce them must include 4 key phases, including the study of feasibility, elicitation – analysis, the specification and the requirements validation [132], [8]. To achieve sufficient clarity of the System Requirements, Sommerville also suggests a standardized form of the latter with the inclusion of a short rationale to explain the necessity [132]. A standardized outline of the Software Requirements Specification (SRS) is presented by the IEEE – 830 – 1998 standard [73] in order to promote the key characteristics for an SRS document of high quality. According to the Standard, these characteristics are:

- Correctness
- Unambiguity

- Completeness
- Consistence
- Ranking for importance or stability
- Verifiability
- Modifiability
- Traceability

The expression of the SRS is crucial for the assurance of the above key features. The usage of structured natural language is a way of maintaining a standardized form of the requirements reducing disambiguities, whilst maintaining the expressiveness and understandability of natural language. Another method of requirements presentation is the formal system specification. This is based on formal methods, which are derived from mathematical representations [132]. There have been extensive arguments amongst software engineering researchers about the effectiveness of formal methods. These are often used in critical safety systems, although it is argued that the achieved level of their correctness is already high without the necessity for formal methods to be applied [21]. According to Broy (2009) [21], the following principles apply in software development:

- Separation of Concerns
- Stepwise refinement
- Modularity and Compositionality
- Decomposition
- Abstraction
- Rigor and formality
- Generality
- Mitigation of risk
- Anticipation of Change
- Incremental Development
- Standardized Patterns
- Scalability

From the above, formal methods may contribute to the first 6, whereas the last 5 are not addressed by these methods [21]. Although formal specifications may uncover problems and ambiguities, they require a very detailed analysis and high cost to be developed [132].

3.5.3 Architectural design

The architectural design of a software system is determined by the character and the functionality of the product, as it affects the performance, the robustness, the maintainability and the distributability [16]. Software architecture has been evolving during the years with a purpose to assist developers to overcome common design issues and problems such as correct communication and communication of developers working from different divisions or reusing components. It represents the abstraction of system construction patterns [125]. Several architectural styles have been developed. The repository model is convenient for an integration of sub – systems with large amount of information exchange. The client server model can be applied for systems that involve clients requesting remote services. This model has the sub systems organized into a group which provides services and into another which requests services. It has a very important advantage due to the distributed architecture [132]. The layered model organizes the system into layers, each of which provides a group of services. It supports the incremental development and it is also changeable and portable [132]. The design patterns introduced by Gamma et. al (1994) [49], is a concept inspired by classical architecture and utilizes standard abstract software design patterns as guidelines for developers (Figure 24). Software design descriptions are also standardized by the IEEE Standard for IT systems design descriptions [71], which provides guidelines for selection of the appropriate design description language.

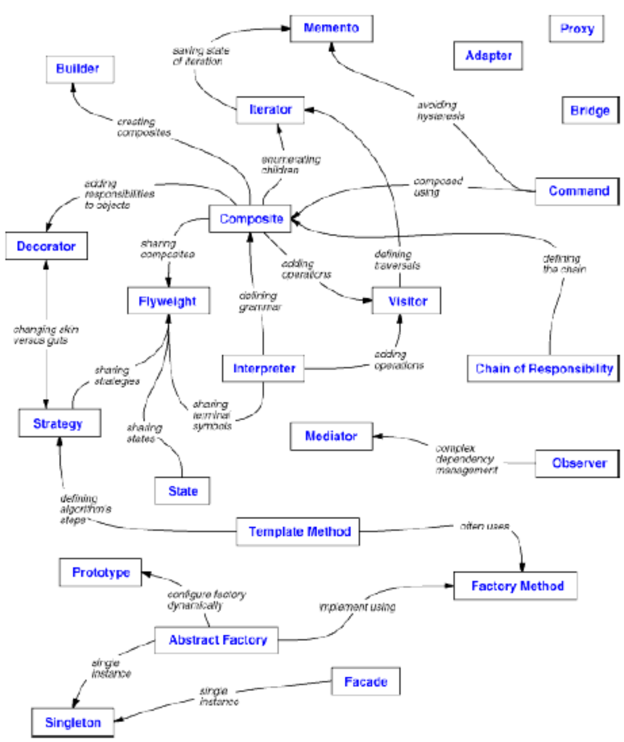


Figure 24 – Design patterns and their relationships [49]

3.5.4 Validation & Verification

The consensus of the field in software engineering is that during and after the implementation process, the developed program should be checked to ensure that it properly meets the requirements and functionalities it was designed for, prior to delivery [132]. The checking processes used for this purpose are known as verification and validation [132]. Simply, Boehm (1984) states their difference [15]:

- 'Validation: Are we building the right product?'
- 'Verification: Are we building the product right?'

According to Sommerville (2006) there are two complementary approaches in the V&V process: software inspections, which are static analysis of the SRS document, the design document and the source code, and testing which is a dynamic approach based on test data [132]. The various methods which may be applied depend on the software process model. In a waterfall model the product cannot be tested before completion whereas in incremental or evolutionary approach, testing may be applied in between stages. Based on the definitions and guidelines of the DO-178 document, NASA [65] presents the types of structural coverage required for avionics software validation, based on control flow of the software. According to DO-178B, table A-7 of Annex A9 [65], Modified Condition/Decision Coverage is required for software level A, Decision Coverage for levels A-B and Statement Coverage for levels A-C. As required by DO-178, the control flow testing takes place after the requirements testing, in order to exercise code statements and decision nodes not invoked during the requirements testing.

Research for improvement of testing strategies is ongoing in order to improve testing capabilities in contemporary complicated software systems. Hayes et al. propose the usage of fault links (i.e. relationships between the types of code faults, or defects, and the types [64] of components in which faults are detected) in order to augment code reviews [64] (Figure 25). Clarke addresses the problem of testing abstract classes (as abstract they cannot be instantiated and tested independently with the use of standard testing strategies), by presenting a structured approach focused on ensuring the test of the features defined in the abstract class. The suggested strategy consists of first classifying the features of abstract classes and then identifying truly inherited methods [29]. Yoo et al (2012) introduce an optimizing approach to automated software test data generation for application in structural testing. Their work is based on the assumption that some data already exist and generates test data by exploiting the existing data [162]. Finally, changes imposed on a software product may harm its established behaviour. A particular strategy for verifying that changes do not affect the product negatively is regression testing. The importance of this strategy is stressed by a survey published by Yoo et.al (2012), where minimization, selection and prioritization techniques are presented, open problems are discussed and potential directions for future research are proposed [162].

Figure 25 - Fault code taxonomy [64]

3.5.5 Model checking

Model checking is a formal verification technique [11] and it is commonly applied in critical software systems. In contrast to simulation, formal verification methods do not rely upon the dynamic response of the system, but perform static analysis on a formal mathematical representation, in order to check it for correction with respect to a given specification [4]. It is also considered that the integration of formal methods - such as model checking - into software development environments contributes to the fight against increasing cost and complexity with automation and rigour [30].

A practical implementation of model checking in critical software systems may be seen in an industrial application, described by Cimatti et. al [28], where the formal verification of Safety Logic of an interlocking system for the control of railway stations - developed by Ansaldo - is presented. The model checking is accomplished with the use of SPIN, one of numerous related software tools. The application of model checking is also becoming more favourable with the DGS-2100, a Rockwell Collins product that provides the cockpit displays and display management logic for large commercial aircraft. Rockwell Collins has developed a translation framework that bridges the gap between some popular industrial model-based development languages such as ESCADE or LUSTRE and several model checkers [30]. Specifically, the translation framework has focused on NuSMV, another model checking tool, developed by ITC-IRST, Carnegie Mellon University, the University of Genoa and the University of Trento.

There are several model checking tools available, namely SPIN, NuSMV, FDR2, CADP, ALLOY and PROB [45]. Each tool has a set of properties and standards that characterize it and different tools might be more suitable for some cases than others. For example, SPIN is an explicit model checker and thus uses an explicit representation of the transition system associated to a model specification, whilst NuSMV is a symbolic model checker and represents the transition system as a Boolean formula. However, they both support temporal languages like LTL (Linear Temporal logic) and CTL (Computation Tree Logic) for property specification [45]. Temporal logic is a special case of modal logic, which allows the reasoning of the evolution of predicates along time. LTL and CTL have different path quantifiers and even those with the same meaning cannot always construct equivalent formulae. This fact indicates that the two logics have different expressing powers [146] (Figure 26). However, as stated by Vardi, M. (2001) [142], verification is considered easier with CTL and preferred in the industrial world against LTL, although the restricted syntax of CTL limits the expressive power of this Logic. On the other hand, LTL is considered not expressive enough [142]. Also, CTL is more exhaustive in the expression of a path existence compared to LTL.

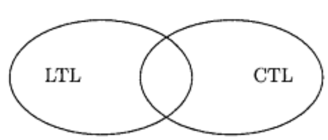


Figure 26 – Relation between LTL and CTL [146]

3.6 Real time operating systems interfacing software

The interfacing of a mechanical component such as a gas turbine with a computer application could be found in real – time operating systems. Suitable programming languages (usually known as synchronous programming languages) for real – time embedded systems should include provisions to access hardware components. Systems – level languages such as C may also be used [132]. A system which responds to sensor stimulus is shown schematically in Figure 27.

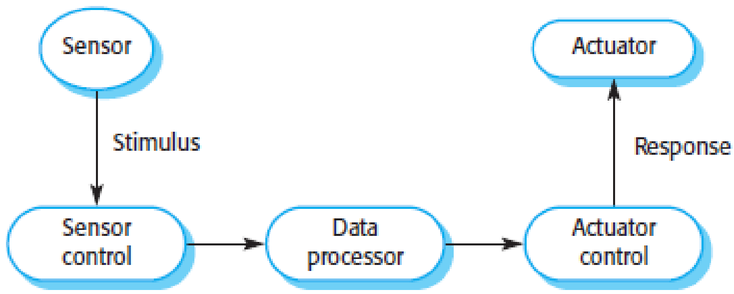


Figure 27 – Sensor/actuator process [132]

It is often that real – time system programming languages appear integrated in development suites. Scade Suite developed by Esterel Technologies is such an example, where scade language, a derivative of LUSTREL is integrated. Scade suite is a model based development environment used for critical embedded software design in aerospace, rail transportation and heavy industry [35].

A widely known programming language often used in real – time systems is MATLAB, developed by Mathworks [91]. It is also integrated with Simulink, which is a development environment for multi-domain simulation and model-based design for dynamic and embedded systems [91]. MATLAB includes a wide range of toolkits which enhance its capabilities. With appropriate toolkits, MATLAB can communicate with data acquisition devices (DAQ's) of third party designers. Simulink can be translated and execute C or C++ code with the use of Simulink coder [92]. Simulink however is mostly specialized in the design and modelling of control systems.

Another familiar tool suitable for real – time applications is LabVIEW, a graphical environment developed by National Instruments which integrates programming language G [100]. This is more specialized in data acquisition and controlling devices. It is also accompanied with compatible hardware components manufactured by National Instruments hence allow easy interfacing of mechanical devices. LabVIEW can communicate with native or higher level languages (C++, Java or .NET applications) via shared libraries (DLL) [106] or by TCP connection with the implementation of appropriate visual instruments, as Ko et. al demonstrate in their application of a web based laboratory for control experiments on a coupled tank apparatus real – time control system in distance learning [84]. Another way of calling LabVIEW from other languages is by using Microsoft ActiveX [105]. However, according to developers' opinions in National Instrument forums [107], access via DLL is considered a faster technique rather than via ActiveX. Java can communicate with LabVIEW with prior conversion of Java code into native with the use of appropriate tools such as Java Native Interface (JNI) or Java Native Access (JNA).

3.7 Internet applications and protocols

Preparation of an Internet application requires certain knowledge about the existing protocols applied to communicate data and the tools available in order to construct the most efficient and appropriate solution.

3.7.1 Internet data communication protocols

Protocols describe the forms of transferring transfer through the network. According to the International Organisation of Standardisation, the network protocol layers are described by the Open Systems Interconnect (OSI) model [74] (Figure 28). The Internet layer model (Figure 29) does not show a one to one correspondence to the OSI model and has been developed based on the necessity and by numerous people co – operating loosely.

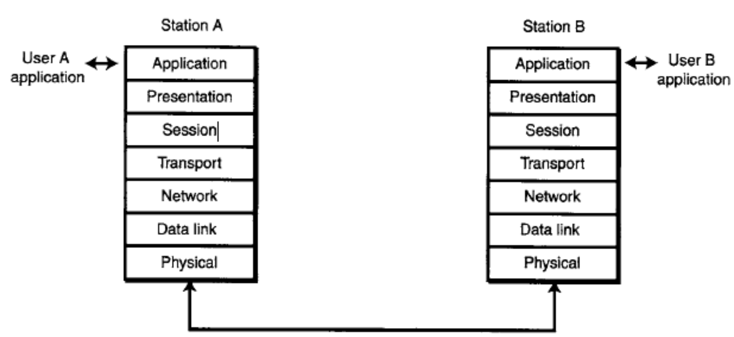


Figure 28 – The OSI model [74]

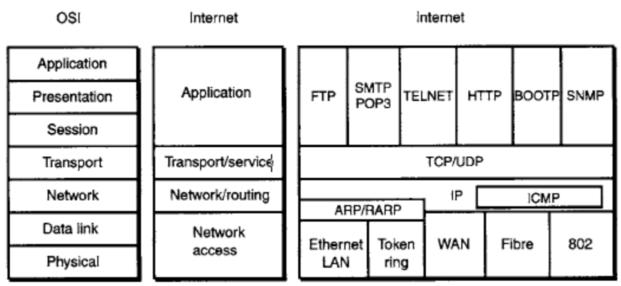


Figure 29 – The Internet layer model [74]

3.7.2 Web services and web applications

A Web service is a means of providing operations to remote clients using standard Internet technology. According to the World Wide Web Consortium (W3C), 'A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.' [143].

It is noticeable that Web services typically use HTTP protocol for the application layer. HTTP, currently upgraded to HTTP/1.1 [150], is a basic communication protocol on the Internet, suitable for transferring images, sounds or several different types of files. It applies a simple client – server methodology and all the requests are handled individually. Most web browsers support all versions [74]. HTTP runs on top of TCP at the transport level, although it alters some of the TCP methodologies. The main contradiction between them is that TCP is session oriented and maintains a logical connection between the client and the server for the time of the data exchange whilst once HTTP conveys a message; the two terminal machines forget each other [9]. One more interesting feature of HTTP is that the HTTP binding, as defined by W3C, natively supports the Request – Response and the SOAP Message Exchange Patterns, thus utilizes the capabilities of the underlying protocol in order to implement an abstract feature [61].

Apart from XML web services described above, there is another approach to remote services known as RESTful web services. These do not adhere to any particular web service protocols but represent an architectural design of services, abstracted from HTTP 1.1, as defined by Fielding (2000) [40]. Web sites, images or CGI scripts are considered as resources that can be addressed using URLs and a web application is an accumulation of resources. With HTTP requests, messages may be sent to the resources and a direct manipulation of a resource is not intended. Each access is made indirectly using a URL assigned to the particular resource [122] Figure 30.

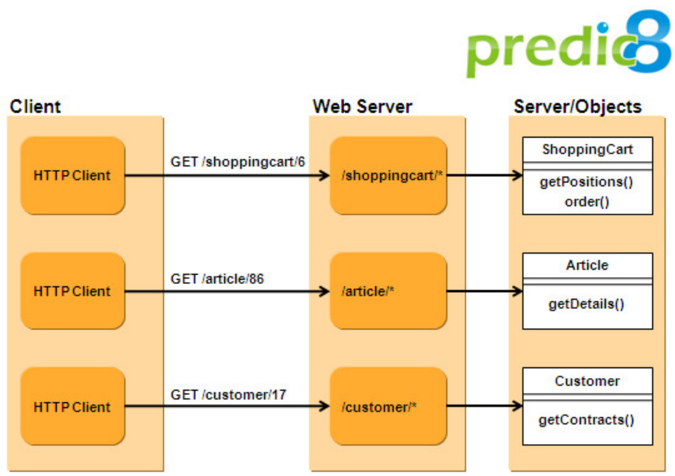


Figure 30 – RESTful web services [122]

According to Fielding (2014) [39] not any HTTP based interface may be called a REST API, unless it adheres to several strict rules, such as communication protocol independence, no alteration of communication protocol, no definition of fixed resource names, no typed resources significant to the client and 'A REST API should be entered with no prior knowledge beyond the initial URI (bookmark) and set of standardized media types that are appropriate for the intended audience'. This definition has caused a wide debate about what should be called REST API and what RPC, which is an approach more relevant to XML web services.

The term STREST (Service Trampled REST) [127] is being used to describe services claiming to be REST API's but do not adhere to the strict guidelines of Fielding. However, Richardson (2008) [127] disputes the term STREST and describes these services as REST – RPC hybrid web services. According to Richardson (2008), the latter are considered to have architecture in between REST and RPC. They use HTTP as a convenient envelope in a manner that overlaps with what a RESTful web service might do and there are occasions that this hybrid architecture can be used to form powerful web tools.

Pautasso et. al. (2008) [119] presented a study that compared "big" (XML) web services and REST. In their study, there is a discussion with regards to differences in protocol layering between the 2 approaches, and the debate takes place around the question, whether HTTP is considered as an application or a transport protocol. Their observation states that in RESTful services, the Web is seen as the universal medium for publishing globally accessible information, thus HTTP is used as a uniform application protocol. Applications become part of the Web by using URIs to identify the provided resources, data, and services and by leveraging the full semantics of the 4 HTTP verbs (GET, POST, PUT, and DELETE) to expose operations on such resources. By contrast, in XML services, the Web – thus HTTP - is seen as the universal transport medium for messages, which are exchanged between Web services endpoints of published applications with only one HTTP method (POST).

The main conclusion from the study of Pautasso et. al. (2008) [119] was that the tool styles have many similarities. REST was regarded efficient in terms of flexibility and control, yet requires a lot of low-level coding. XML web services provide better tool support and programming interface convenience, but introduce dependency on vendors and open source projects. Their main recommendation from this comparison was to use RESTful services for tactical, ad hoc integration over the web and to prefer XML services in professional enterprise application integration scenarios, with a longer lifespan and advanced QoS requirements.

3.7.3 HTTP requests

An HTTP request is a message package from the client to a server. It contains standard lines of information, one of which depicts the request method to be applied to the end resource [41]. Considering 2 of the most commonly used http request methods, GET and POST, the following can be noted [152]: GET is used to request data from a specific resource whilst POST is applied to submit data to be processed to a specified resource. Noticeable differences are shown in Table 2.

Table 2 – Differences between GET and POST methods [152]

GET	POST
requests can be cached	requests are never cached
requests remain in the browser history	requests do not remain in the browser history
requests can be bookmarked	requests cannot be bookmarked
requests have length restrictions	requests have no restrictions on data length

3.7.4 File Transfer Protocol

FTP (File Transfer Protocol) was designed for transmitting files and fixed data blocks. It is used to upload/download applications or data to or from a local machine or a server [61]. It opens 2 ports and runs on top of TCP as well. Although it requires the opening of 2 TCP connections, transactions may be interleaved. The requirement for 2 ports poses a burden to the application compared to HTTP and also command transmission is more complicated than in HTTP [74].

3.7.5 TCP and UDP

It can be inferred from the previous sections that TCP (Transmission Control Protocol) protocol of the Transport layer acts as the main mean for transmission under HTTP and FTP. However, UDP (User Datagram Protocol) may be preferred in some cases instead of TCP. TCP is considered as a reliable connection [74] and requires acknowledgment between sender and recipient. The shortcoming of this protocol is that it is relatively slow [74]. By contrast, UDP does not apply acknowledgement but transmits data faster and it is used by application layer protocols such as RTP which transmits and receives real time data, including voice or image [74]. UDP is not as reliable as TCP (which additionally delivers error messages) and should require additional work for the developer in order to implement reliable communication on top of UDP. However, the additional work in combination with lack of experience could lead to failure prone connections.

3.8 Performance and security over the web

The Internet may constantly be developing, providing increasing applications and ease of use, but also, the threats that accompany it rise and become more and more difficult to counteract.

3.8.1 Internet threats

A system once placed on the Internet becomes vulnerable to exposure. Classification and quantification of the Internet threats is not easy to define. According to Vacca (2007) [141], a general classification of threat types can be presented as shown below.

- Hackers. This group of attackers aims to break into computer systems and exploit them for various reasons.
- Malware. Disguised pieces of code causing undesirable events transmitted with disks, email or other communication means.
- Spam. Unsolicited commercial email messages.

- Denial of Service (DoS). During this attack the perpetrator renders an enterprise unable to use certain network resources by introducing network traffic which exploits an existing weakness of the system.
- Inappropriate Web Usage. This implies the access of inappropriate material from users or usage of the network to accomplish personal matters.
- Insider Attacks. Either disgruntled employees or others aiming to financial gain and access privileges can pose an interior threat to a system.

Vacca (2007) [141] also states that the volume of Internet attacks increases and the nature of the attacks becomes more sophisticated. Assessment of the threats is suggested by the cost of the damage to the enterprise. Prevention of the threats requires extensive knowledge of the nature and the purpose of the potential attackers, motives and target Vacca (2007) [141].

3.8.2 Internet security

Various ways of protection against threats may apply according to each specific application and ongoing research aims to optimisation of techniques for their identification and mitigation. Roy et. al. (2012) [128] have presented a novel approach known as attack countermeasure tree (ACT) which takes into account attacks as well as countermeasures and avoids the generation and solution of a state-space model in its analysis. Three case studies have been used to study the results of implementation of countermeasures (ACT for BGP attack, ACT for a SCADA attack and ACT for malicious insider attacks) [128]. A firewall placed at an appropriate station may control the traffic of the network. Incorporation of application proxies can block several forms of attacks disguised as legitimate traffic and perform other security and inspection functions. Firewalls can also be used to segment internal networks [141]. Malware protection can be obtained with the installation of host – based or gateway – based scanners to search targeted software components against known threats [141]. The Virtual Private Networking (VPN) is a mean of establishment of secure virtual tunnels through the Internet. It provides the security benefits of private lines with the cost structure of public networks [141]. Wade et. al. (2010) [144] proposed a cross – layer approach to mitigate distributed DoS, by implementing a remote firewall, or a device driver packet filtering. The concept of the remote firewall aims to protect access links with limited bandwidth from being overwhelmed by DDoS traffic and flash crowds. As described by Wade et. al (2010), filtering malicious and excessive legitimate traffic at a local firewall may be too late if a surge in network traffic clogs the access link. Thus a remote firewall drops or rate-controls such as potentially harmful traffic before they get in the access link. Furthermore, the remote firewall is proposed to allow administrators to update their firewall configurations without being dependent on their Internet service providers and consequently reducing administrative delay. The second approach (packet filtering) is achieved by implementing two Bloom filter based packet filter methods and has the advantage that the malicious packets are dropped at the earliest possible time before they are processed by the upper layers in a protocol stack. Hence, consumption of processing a resource of a victim server by the malicious packets is prevented. As a conclusion of the research, it was stated that device based packet filter implementation has shown satisfactory results, whilst remote firewall can provide additional mitigation of DoS on top of other applied solutions [144].

Installation of switches between the hardware components of a computer system may enhance security, as they provide an additional level of security, rendering the tracking of a network more difficult [46]. Intrusion Detection Systems (IDS) have been developed to monitor computer systems or networks for unauthorized access and misuse. These systems can be considered as a second line of defense against attacks on networks and are becoming more popular as the likelihood and the severity of potential attacks increases [31]. According to Di Pietro (2008) [31], they can be classified in signature based and in anomaly based systems.

Several techniques of encryption of the data transferred are available. Authentication and access management is a highly important element of assuring security of a software system over the Internet. Restriction of access to unauthorized users does not eliminate, but seriously minimizes the risk of allowing potential attackers to penetrate. Wilding (2003) [149] stresses the vulnerability of static passwords used for authentication and addresses the necessity more secure methods for encryption of electronic fund transfer systems or other forms of high security processing. In Wilder's password disclosure matrix there are more than 10 ways of password cracking displayed. Consequently, the authentication method applied contributes to the security of the credentials transmission. A form based authentication, although not easy to implement, may easily encrypt the data with the use of HTTPS, in contrast with other less secure methods, such as the basic authentication, where passwords are not encrypted [66]. Another even more secure technique is the client side certificate issued by appropriate authorities. However, this method is not convenient and increases the cost of the application [66].

Increase of security may be obtained with the implementation of a secure protocol in the application layer that allows for data encryption and user authentication. An example is the Secure Shell protocol, SSH, offered by the SSH Communications Security Corporation [75]. However, compatibility with other security targeted applications needs to be taken into account. It is noted here that commercial SSH presently does not implement VPN. To combine SSH and VPN the designer should use the openSSH, which is a free version of SSH [110].

Another related factor to be considered is the contribution of the HTTP request methods applied to the security of transmitted data. In the HTTP specification, it is recommended that GET method should not be used to transmit passwords or other sensitive data with HTTP, as this will cause this data to be encoded in the Request-URI. There are many cases of servers, proxies and user agents that may log the request URI somewhere visible to third parties [42]. POST-based form submission can be used instead, where parameters are not stored in browser history or in web server logs [152]. Although storage and logging restrictions in POST method may increase the security level against GET method, in order to achieve encryption of the transmitted data in both methods, the implementation of a secure transport layer would be necessary (HTTPS, running on top of SSL).

Finally, the inside attacker cannot be taken lightly. This type of threat should always be considered as possible, even if such a possibility may be remote, depending on the enterprise internal relations between staff and administration or the internal level of physical security. You et. al. (2012) have recently published a proposal for defense against insider threats and internal data leakage [163].

3.8.3 Network performance

Apart from security, the other major concern of an Internet application is Quality of Service (QoS), which is closely related to the network utilization. Network utilization is defined as the ratio of demand over capacity of the network and has direct effects to performance [31]. According to available literature, 30 % of utilization is a commonly cited rule of thumb for network utilization [70]. TCP provides congestion control and ensures reliable data by retransmitting unacknowledged segments, which however may be a problem for several real time applications, including sound and voice transfer. As an alternative, UDP may be used but this protocol does not provide congestion control [70].

Performance of a network may be described by several parameters. Halsall (1992) [63] presents an example of performance measurement of a Local Area Network (LAN), where the mean transfer delay of a frame of data to be transferred across the network is considered, against the normalized throughput, which is the ratio of the offered bit load over the available bit rate [63]. However, a short finite time delay is always expected during data transmission, as a result of the physical impedance of the medium connecting the ends and the distance between them. This is known as the transmission propagation delay and may be described by the round trip delay, which can be defined as the time delay between the first bit of a block transmitted by the sender and the last bit of its associated acknowledgement being received [63]. This parameter is a function of the time for a frame to be transmitted at a given link bit rate and the propagation delay, which depends on the type of the medium and the distance. As an example given by Halsall (1992), the signal propagation through a twisted pair wire or a coaxial cable for 1 meter length will last 0.5 x 10^{-8} sec.

Corruption of data during transmission is very likely to occur, especially when long distance separates the ends or noise is present. For improved performance, error control is introduced as the cycle of combined error detection and correction. Once the errors are detected, the assurance of the transmitted data correction is required [63]. Halsall (1992) [63] refers to two approaches for this process, the forward error control and the feedback error control. In the former, the character or frame transmitted contains redundant information to allow the receiver to detect when and where errors are present. In the latter, the transmitted frames only inform the receiver about the presence of errors but not their location. Various detection schemes may be used and these are determined by the bit error rate and the type of errors. Namely, a few error detection methods are the parity bit method, commonly used with asynchronous and character oriented synchronous transmission, the block sum checked, suitable when blocks of characters are transmitted and the cyclic redundancy check when error bursts are present [63].

Another important element related to data link protocols is flow control. This parameter describes the control of the transmission rate of frames or characters in a way that the receiver always has sufficient buffer storage in order to accept the data before processing them. There is an automatic flow control facility often invoked, which involves the return of special control characters X-OFF and X-ON. On reception of the former, the keyboard ignores the input of additional characters, until the reception of the latter. Another mechanism present is the sliding window, which is suitable for the control of frame flow across a link. Briefly, this mechanism allows the transmission of new frames only after acknowledgement of reception of the previous [63].

An error and network condition reporting protocol integrated with the IP implementations is the Internet Control Message Protocol (ICMP). ICMP includes a variety of messages that describe the condition of a network allowing for network management. Basic functions related to this protocol include error reporting, reliability testing, congestion control, route – change notification, performance measuring and subnet addressing [63]. ICMP detects IP datagrams (format of an IP data unit) discard by a host or gateways of a network and issues appropriate messages. A software tool associated with ICMP messages for checking connection of remote servers is PING (Figure 31). It implements the Echo request/reply function of the ICMP to verify the reachability of a host or gateway and it may be executed as a command in different operating systems. However, it must be noted that as a security measure, many organizations may block the return of ICMP package, preventing the function of PING [95].

Function	ICMP message(s)	Use
Error reporting	Destination unreachable	A datagram has been discarded due to the reason specified in the message
	Time exceeded	Time-to-live parameter in a datagram expired and hence discarded
	Parameter error	A parameter in the header of a datagram is unrecognizable
Reachability testing	Echo request/ reply	Used to check the reachability of a specified host or gateway
Congestion control	Source quench	Used to request a host to reduce the rate at which datagrams are sent
Route exchange	Redirect	Used by a gateway to inform a host attached to one of its networks to use an alternative gateway on the same network for forwarding datagrams to a specific destination
Performance measuring	Timestamp request/reply	Used to determine the transit delay between two hosts
Submit addressing	Address mask request/reply	Used by a host to determine the address mask associated with a subnet

ICMP = Internet control message protocol

Figure 31 – ICMP messages and their application [63]

4 SOFTWARE PROCESS, REQUIREMENTS AND ARCHITECTURE

This chapter presents the methodology applied for the overall software process along with the methodology for the selected architecture. It also describes the definition of the System Requirements, the System Requirements Specification and the validation of the requirements.

The system was developed following an evolutionary process, leading to the design of 3 distinct prototypes: Operation by local PC, Operation in LAN and finally Internet operation

4.1 Software Process

4.1.1 The Spiral as the Software Process Model

The software process was selected to adhere to an adaptation of the Spiral Model. This was introduced by Boehm (1988) with the purpose of overcoming the difficulties imposed by the existing up to the then-prevalent techniques, such as the waterfall model, the evolutionary and the transform model. It had been based on experience with various refinements of the waterfall model as applied to large government software projects [14]. According to Boehm, the radial dimension represents the cumulate cost incurred in the accomplishment and the angular dimension depicts the progressed achieved in the completion of each cycle (Figure 32).

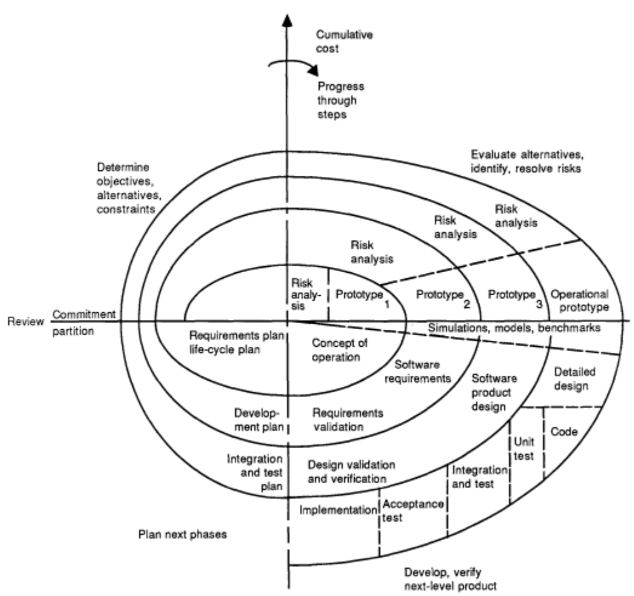


Figure 32 - The Spiral Model as introduced by Boehm [14]

Each cycle of the spiral is initiated with the identification of: the objectives of that cycle, the alternative means of implementation and the constraints imposed by the alternatives, with risk analysis to follow. At an example provided by Boehm (1988) [14], indicatively the rounds of a spiral process may include the following steps:

- Round 0: feasibility study
- Round 1: concept of operations, including the key points of objectives, constraints, alternatives, risks, risk resolution, risk resolution results, plan for next stage, commitment
- Round 2: top – level requirement specification with consideration of the respective key points of this stage, as in Round 1.
- Succeeding rounds: development, Implementation and validation

Several features of the spiral identified from the examples of Boehm are:

- Fostering of the development of specifications that are not necessarily uniform, exhaustive or formal
- Incorporation of prototyping as a risk reduction option at any stage of the development
- Accommodation of reworks or returns to earlier stages as more attractive alternatives are identified or if new risk issues need resolution.

The spiral differs from the other models mainly in the identification and consideration of risk [132]. Evaluated by Boehm himself [14], its main advantage is the risk driven approach in combination with the accommodation of the good features from other process models. These were the 2 main factors that have led to the selection of this approach for the development of the system in this project.

Prototype driven development is a software process model that was also considered. There are occasions when requirements are not identified with detail by the customer, or the developer may not be sure about the efficiency of an algorithm or the adaptability of an operating system. In these situations, the application of the prototype paradigm (Figure 33) may offer a sufficient solution. The process commences with communication in order to identify any available requirements and proceeds to the quick design and construction of a working prototype application. This model can be used as a stand-alone process, but it is commonly used within other software process models, often for the derivation of more specific requirements [123]. In this project, a prototype driven process would not be sufficient on its own, due to the necessity to consider the risk management, as the project has been regarded as a safety critical system. However, it would be useful in sub phases of the Spiral process and it is used for the requirements validation. It would also be applicable to the design of the final individual components of the application, after the initial implementation rounds were completed successfully.

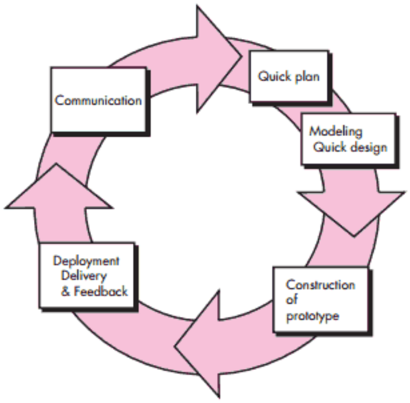


Figure 33 – The prototype paradigm

Agility (or Agile Development) is a software development approach that has been gaining popularity the last years. It adheres to the rules of the Manifesto for Agile Software Development, issued by a group of notable software developers in 2001 [123]. It is not a single standard methodology, but consists of several software development approaches that attempt to overcome the difficulties imposed by traditional process models, such as the waterfall model. They emphasize effective response to changes, facile communication between involved parties and rapid delivery of working software. They de-emphasize the importance of intermediate work products, which is not always considered to be positive [123]. However, in a case of a safety critical application, a well defined (although not exhaustive) initial Requirement Specification should be available, in order to allow for risk consideration, risk mitigation and planning of alternative approaches. Hence agility was not considered as effective as the Spiral Model for this application.

4.1.2 Adaptation of the Spiral to the present project

The rounds of the Spiral in this case represent the following elements (Figure 34):

- Round 0: Feasibility study and SRS
- Round 1: Local Interfacing of the gas turbine – Local PC version
- Round 2: Interaction with gas turbine interface program and design of core operation control program
- Round 3: Implementation of additional performance and safety features – LAN version
- Round 4: Internet Implementation
- Round 5: Acceptance Validation

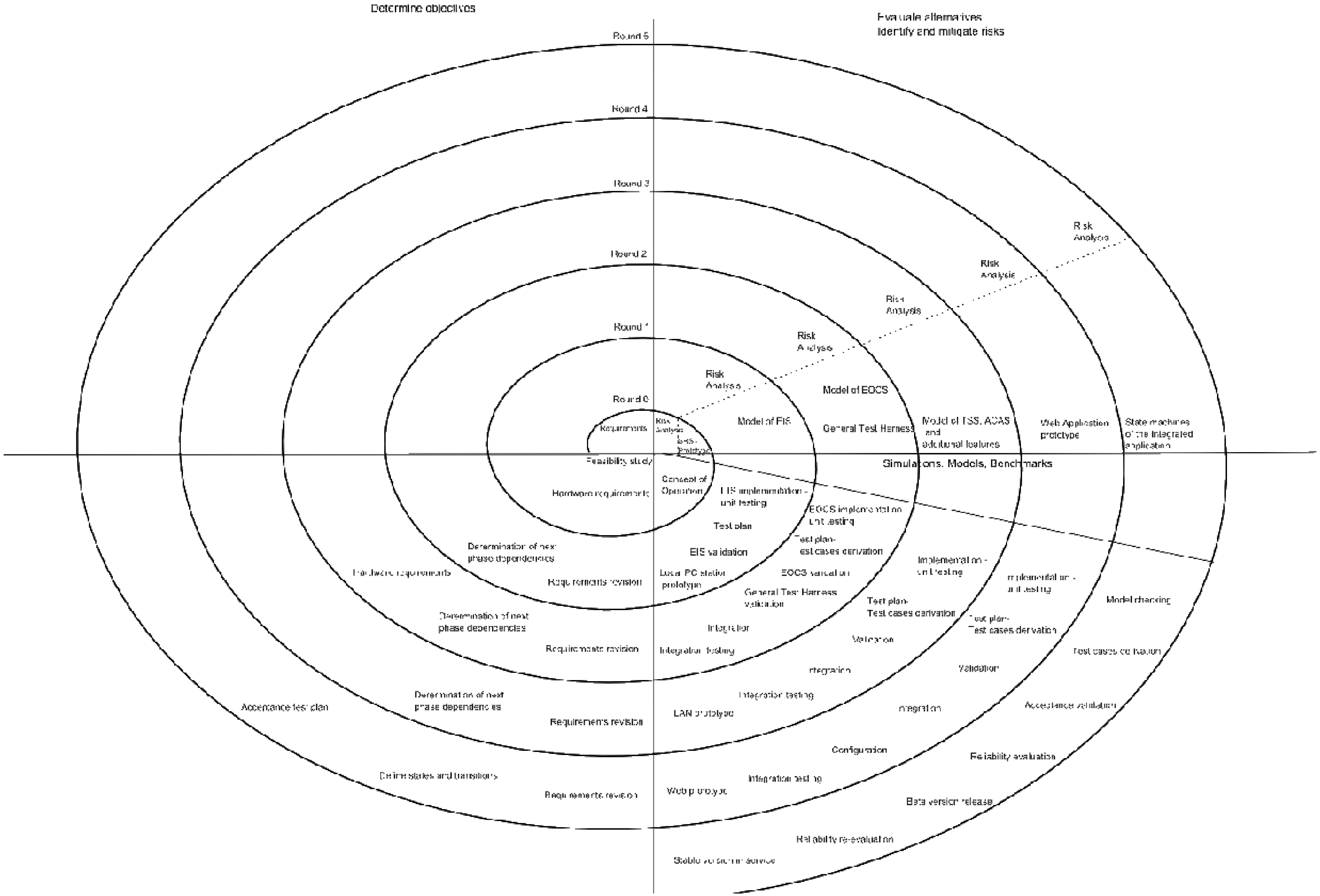


Figure 34 – Adaptation of the Spiral for the current project

4.2 System Requirements

This section describes the methodology applied for the elicitation, definition and presentation of the system requirements. The final version of the Functional Requirements is shown in Appendix A. The requirements have been finalised after the completion of rounds 0 – 3 of the software process, as according to the Spiral Model, requirements are not final from the initiation, but instead, they are revisited after each round and adapted according to risen risk issues and alternatives applied in each round.

4.2.1 Methodology

The general layout of the SRS document adheres to the IEEE – 830 – 1988 – SRS Standard [73]. According to the Standard, key points addressed by the SRS are: the functionality of the software, the external interfaces, the performance, the attributes and the design constraints imposed on the implementation. The document comprises of the following sections:

- Introduction
- Overall description
- Specific Requirements

The Overall Description sub section outlines the features of the system in the form of User's Requirements, statements in natural language that outline what the final system should do [132]. The elicitation of User's Requirements was obtained by collaboration with the involved Departments of the University and has taken into account the risks of operating a gas turbine remotely, the required performance and the prospective of a generic application.

The requirements elicitation process included use – case scenarios derived after discussions with the involved parties. Gallina et. al. (2007) proposed a requirement elicitation template of Dependable Product Lines [48] which includes among others, fault assumptions and risk analysis. Other elements shown and extended by Gallina et. al. (2007) [48], are the description, the synchronous primary actors, resources, dependency, preconditions, post conditions, main scenario, specification of non – functional properties, mis – scenarios and fault variation description. Fault analysis and risk analysis were implemented for the elicitation of the requirements of the project, along with the use of the other aforementioned elements.

Risk analysis forms a basic part of the initiation of the process. The Fault Tree Analysis and Event Tree Analysis were based on the initial use case scenarios [68], [18]. As these methods are recommended by the respective Standard [18] early in the system design stage, they were selected to be applied at this initial stage of the software process, i.e. requirements elicitation, in order to provide a qualitative assessment of risk and introduction of means of mitigation before proceeding further into development. The logical analysis of the FTA was obtained with the implementation of Boolean reduction and the method of minimal cut sets [68], [18]. Two major undesirable events were identified and the analysis was built around them. Moreover, HAZOP analysis was also applied [20], as an assisting tool to allow for direct indication of measures required at the early stages of design. HAZOP was also adapted and applied as a procedural risk analysis for the SRS phase.

The User's Requirements were distinguished in functional and non – functional requirements. The former show the system reaction and behaviour in particular input and certain situations, whilst the latter express various constraints posed on the functions [132]. Functional and non – functional requirements were clearly distinguished and in the SRS each one included a brief rationale in order to justify its existence [132]. The System Requirements were classified in different modes of operation and they were individually numbered. They were expressed in structured natural language, in order to avoid the ambiguities of natural language. The use of formal specification was also considered but as this would add complexity into this time limited project, they were not implemented. Besides, according to Broy (2009) [21], most of the process principles required to be encountered in the SRS are covered with a non formal standardized expression of the system requirements.

The feasibility and validation of the requirements was assessed by review with the collaborating Departments and inspection. Finally, a simple prototype program that simulated the various suggested components of the system and their interaction was designed, in order to render the system and the requirements more understandable. The software components were modelled with the implementation of Unified Modelling Language (UML) [12].

4.2.2 Objectives and risk analysis

In accordance with the Spiral Model, the objectives of Round 0 were defined before proceeding (Table 3).

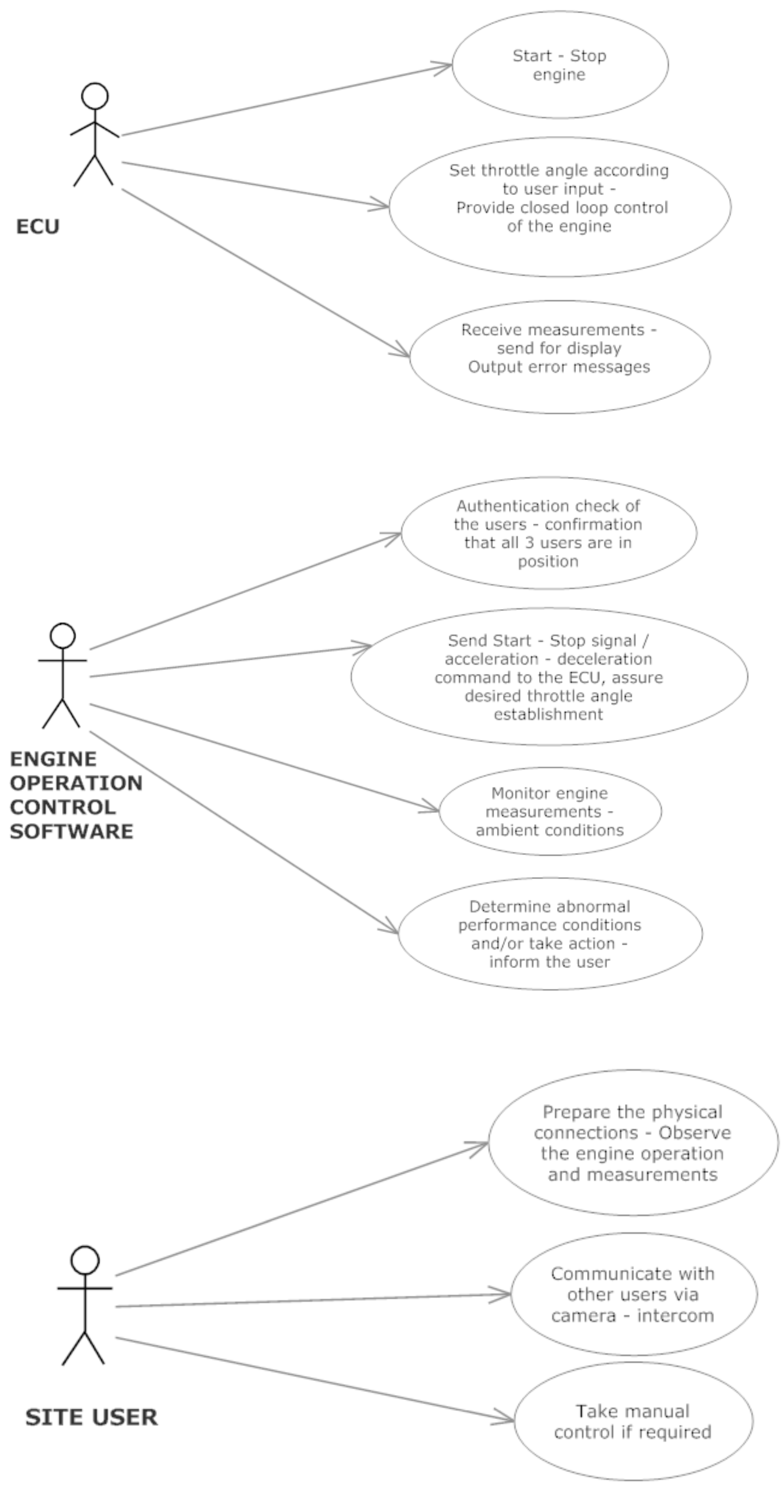
Table 3 - Round 0 Objectives

Elicitation of Requirements
Identify Physical Risks
Define System Requirements Specification
Validate Requirements
Determine Feasibility

Following the definition of the objectives, the procedural risks were identified with the implementation of HAZOP (Appendix B, Table B-2). Significant identified elements with underlying risks included the objectives identification, the feasibility testing of the application, the clarity of the requirements and the identification of the physical risks. It was realised from the very first stages that the requirements should be clearly defined and stated and also the physical risks mitigated prior to further development. The procedural risk assessment dictated the approach for the accomplishment of this round, where risk analysis played a major role. Additionally, cost was identified as an underlying risk, as improper estimation at an early stage would have a negative impact on the overall budget of the project.

4.2.3 Elicitation of the requirements and the SRS

The SRS was derived after elicitation of the requirements, approached by use case scenarios, use case diagrams, collaboration with the involved Departments and consideration of the risks identified in the previous analyses. The use case scenarios were derived with 2 different approaches. First approach with human actors involved (3 users: one in the engine test house, one in the nearby PC station and one through the Internet) and the software represented as a single actor (Figure 35). The other approach was with the hardware and software components as actors, taking into account the abstract definition of the software modules that would be designed later on: The Engine Interface Software, the Engine Operation Control Software and the Client (Figure 36).


ECU
Start - Stop engine
Set throttle angle according to user input - Provide closed loop control of the engine
Receive measurements - send for display Output error messages
ENGINE OPERATION CONTROL SOFTWARE
Authentication check of the users - confirmation that all 3 users are in position
Send Start - Stop signal / acceleration - deceleration command to the ECU, assure desired throttle angle establishment
Monitor engine measurements - ambient conditions
Determine abnormal performance conditions and/or take action - inform the user
SITE USER
Prepare the physical connections - Observe the engine operation and measurements
Communicate with other users via camera - intercom
Take manual control if required

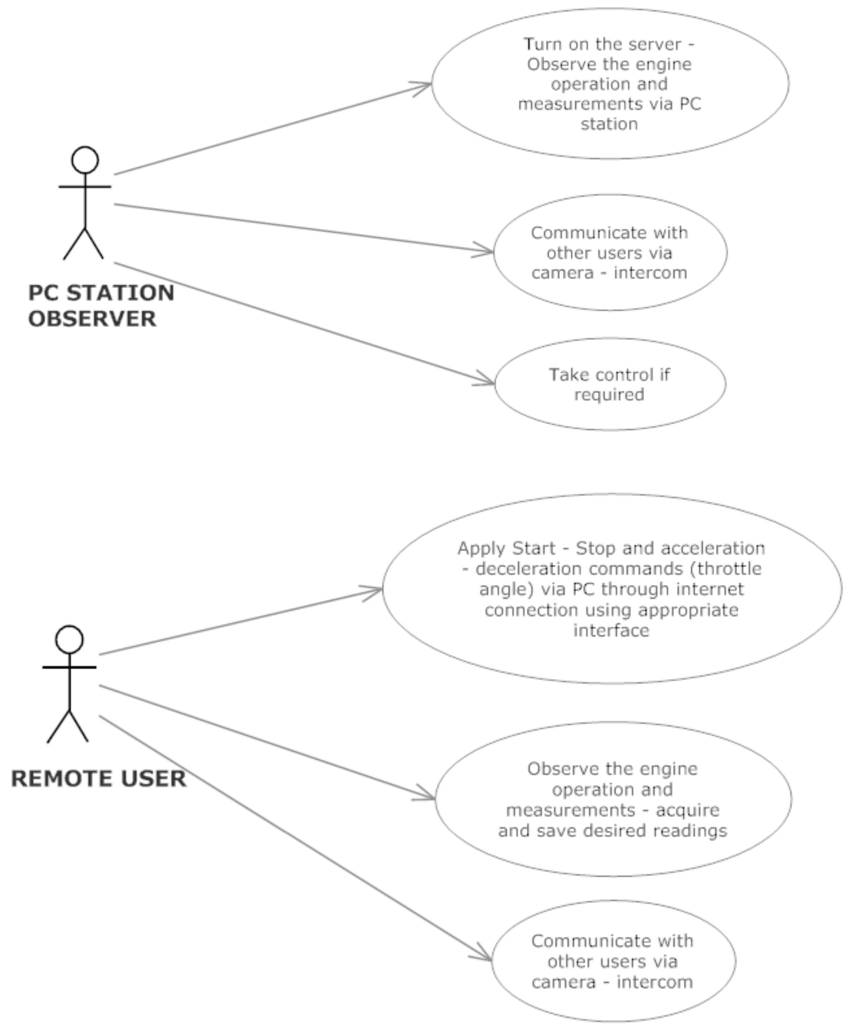
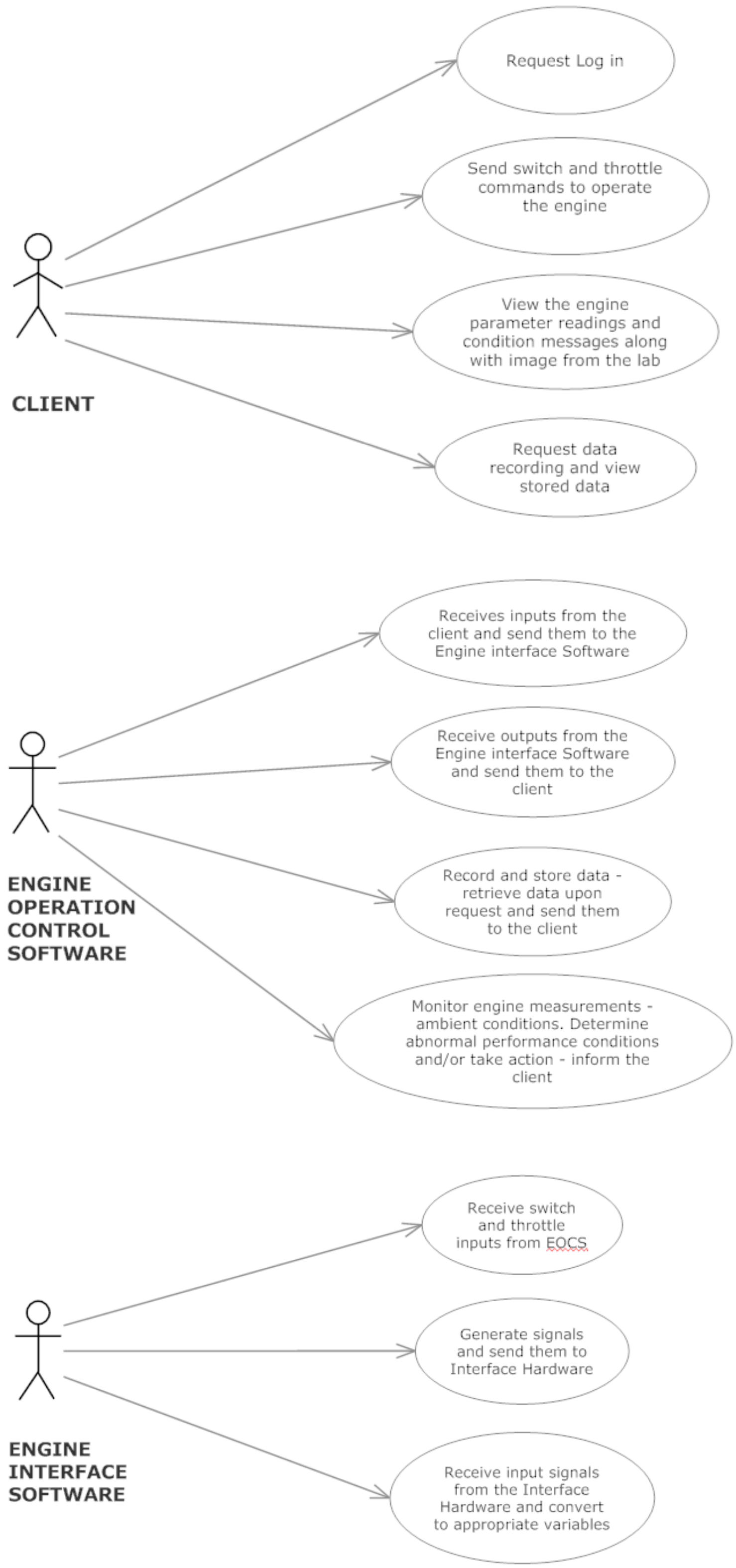


Figure 35 – Use case analysis with human users as actors



Request Log in
Send switch and throttle commands to operate the engine
View the engine parameter readings and condition messages along with image from the lab
Request data recording and view stored data
CLIENT
Receives inputs from the client and send them to the Engine interface Software
Receive outputs from the Engine interface Software and send them to the client
Record and store data - retrieve data upon request and send them to the client
Monitor engine measurements - ambient conditions. Determine abnormal performance conditions and/or take action - inform the client
ENGINE OPERATION CONTROL SOFTWARE
Receive switch and throttle inputs from EOCS
Generate signals and send them to Interface Hardware
Receive input signals from the Interface Hardware and convert to appropriate variables
ENGINE INTERFACE SOFTWARE

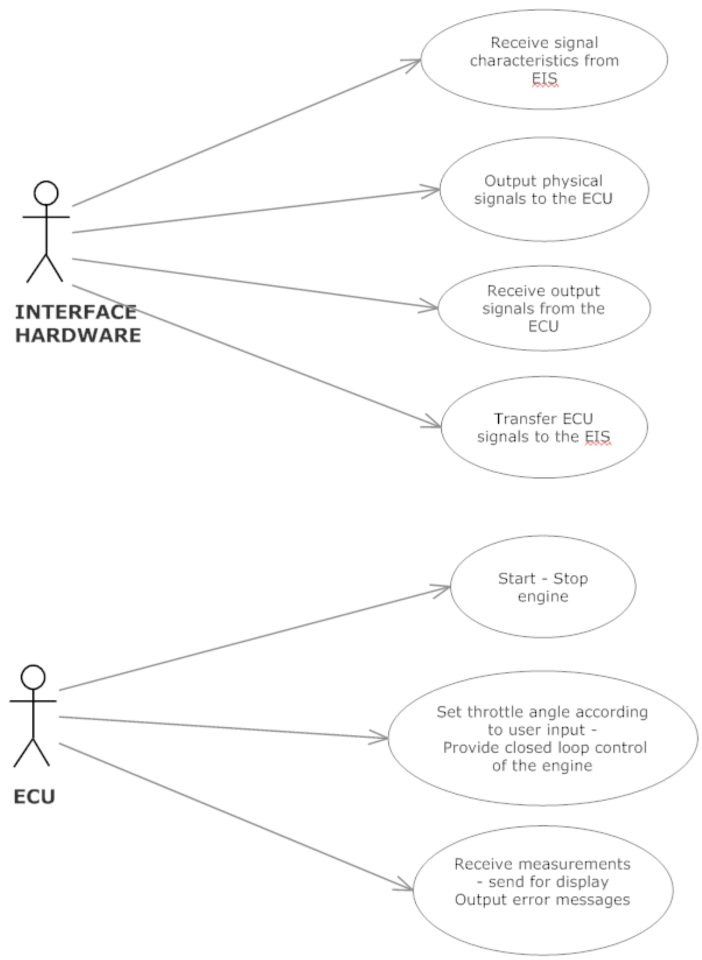


Figure 36 – Use case analysis with software and hardware components as actors

The User's Requirements (Appendix A, Section A.1) were clearly distinguished to functional and non – functional. Key points were:

- The ability of the system to apply automated actions in the case of limits exceedance or power supply and network connection abnormalities. The system shall be used by individuals with basic knowledge of gas turbine operation but due to the variety of experience levels among the users, it must assure that the input is filtered and the engine operated properly.
- Inclusion of provisions for accommodation of generic expansion in the future in order to add other types of gas turbines and interact with external simulation or diagnostics applications.
- It will not be allowed to operate unless the specified observers are in their positions.

Each requirement was expressed with the use of mathematical symbols and notations inspired from object oriented programming. They comprise of 168 individual functions that form the 12 functional requirements, all distributed in 6 basic modes of operation (Table 4).

Table 4 – Functional Requirements

Mode	Description	Functional Requirements
0	Web Application	0.1 Authentication 0.2 System Monitoring
1	Engine start – No malfunction	1.1 Engine Start
2	Engine start – Malfunction present	2.1 Engine Start
3	Engine Running	3.1 Acceleration 3.2 Deceleration 3.3 Parameter Logging
4	Trouble Shooting	4.1 Protection when absolute operating limits exceeded 4.2 Protection when relative operating limits exceeded 4.3 Local Protection
5	Engine Shutdown	5.1 Normal Shutdown 5.2 Emergency Shutdown Local

4.3 Architecture and Modelling

At this point an abstract design of the system was constructed, and the general architecture was depicted, along with the hardware and software components from which the system would be comprised. The system was broken down to multiple sub components - modularity enhances changeability and replaceability; if one of the modules needs to be totally replaced, the impact to the rest of the software will be controlled. An abstract context model (Figure 37) was used herein to represent the sub modules, their interaction and the proposed protocols of communication. The modules were distinguished in 4 major groups: the Olympus gas turbine hardware components, the engine interface software related programs and hardware, the external hardware components and the additional software components required to be designed. The basic components, classified from client's end to service end, were the following:

- The client. Any web browser or a thin client for LAN version.
- Web Application (for Internet version).
- Data Storage System (DSS). Storage of operating information and logs for the engine (future development).
- Main Engine Operation Control Software (MEOCS). Coordinates the interaction between all the different components.
- Mains Supply Interface System (MSIS). Checks the condition of mains power supply to the end server PC.
- Network Connection Monitoring System (NCMS). Checks the network connection between the remote components (future development).
- Ambient Conditions Acquisition System (ACAS). Acquires ambient temperature and pressure.
- Trouble Shooting System (TSS). Receives input from NCMS, MSIS, ACAS and the Measurements Acquisition system and checks the values of the related parameters with regards to predefined limits.
- EIS Interface (EISI). Transmits data between MOCS and EIS.
- Engine Interface Software (EIS). Program designed in LabVIEW to output the function signals to the NI Data Acquisition Unit and read in the measurements.
- Data Acquisition Unit. This is a NI 9174 compactDAQ hardware platform.
- Control Hardware Modules. The NI 9263 C Series analogue output module.
- Measurement Acquisition Hardware Module. Comprises of 2 serial to USB interfaces, an NI 9401 C Series digital I/O module, an NI 9215 C Series analogue input module and a NI 9215 thermocouple input module.
- Junction Box. Comprises of switches and Y – leads to interconnect manual with remote circuits.
- ECU. The Electronic Control Unit of the Engine (Already installed on the engine).
- Manual Engine Control Box Already installed on the engine).
- Engine Data Terminal. Displays the information from the ECU (Already installed on the engine).
- AMT Netherlands Olympus Gas Turbine.

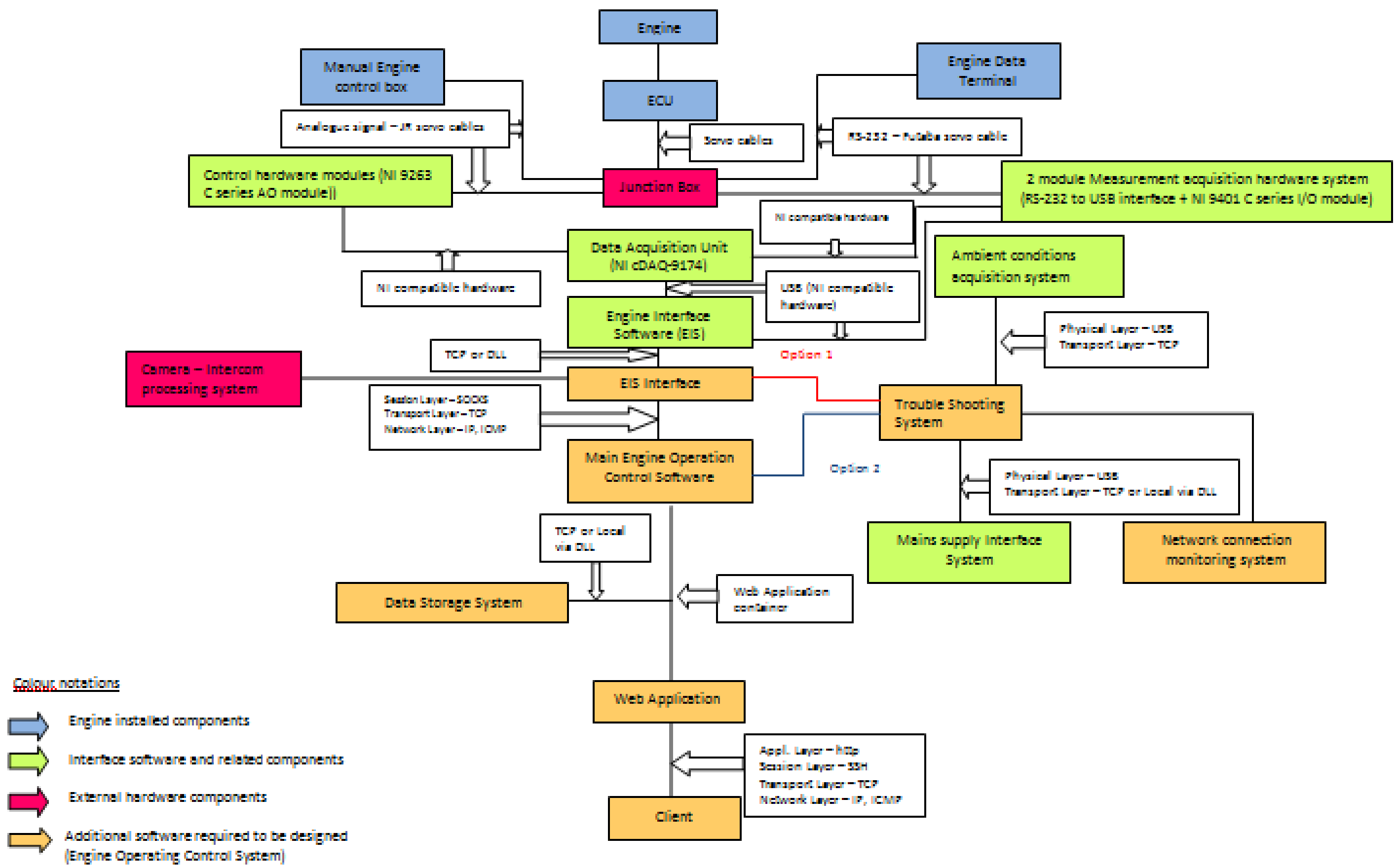


Figure 37 – Context model of the system

The system adheres to the client – server architecture [132]. The repository model would be convenient for sub – systems which exchange large amount of information. Although there is information exchange between modules in the project, their size is estimated to several Kilobytes hence there is no necessity for repository architecture. As the client will request services from the end server the client – server would be more appropriate, also enabling distributed architecture, that also supports the incremental development, changeability and portability, and maintainability [132] (Figure 38).

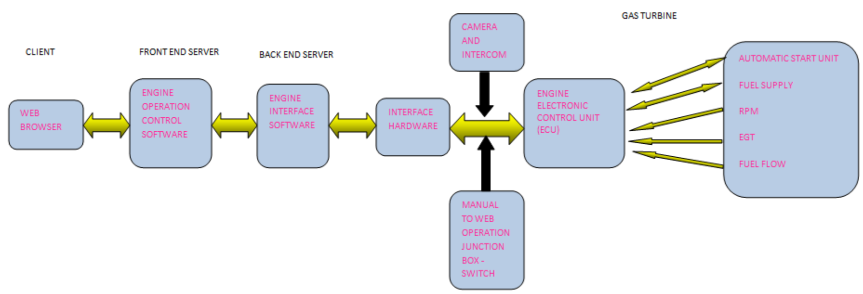


Figure 38 – Architecture and data flow

The various states of the system from engine at rest to start, run and end position, where the engine is at rest again, were distinguished here with the use of an abstract state machine (Figure 39). The sequence of the actions between the various components in response to the different inputs was depicted in a sequence diagram (Figure 39).

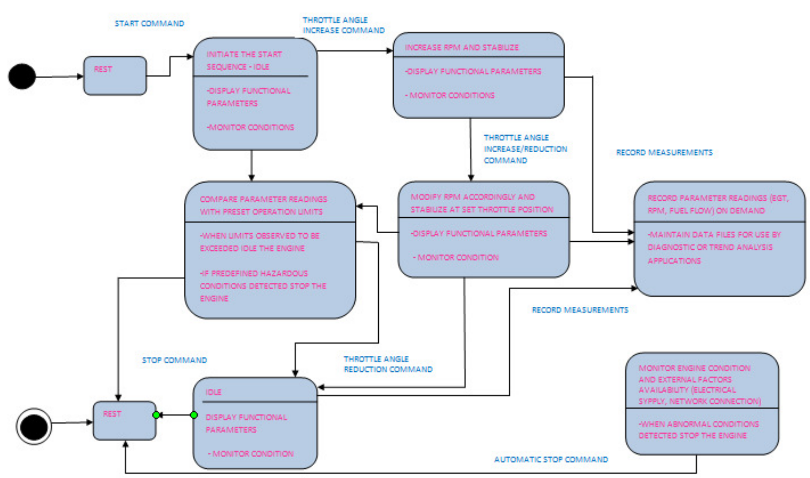


Figure 39 – Preliminary abstract state machine of the system

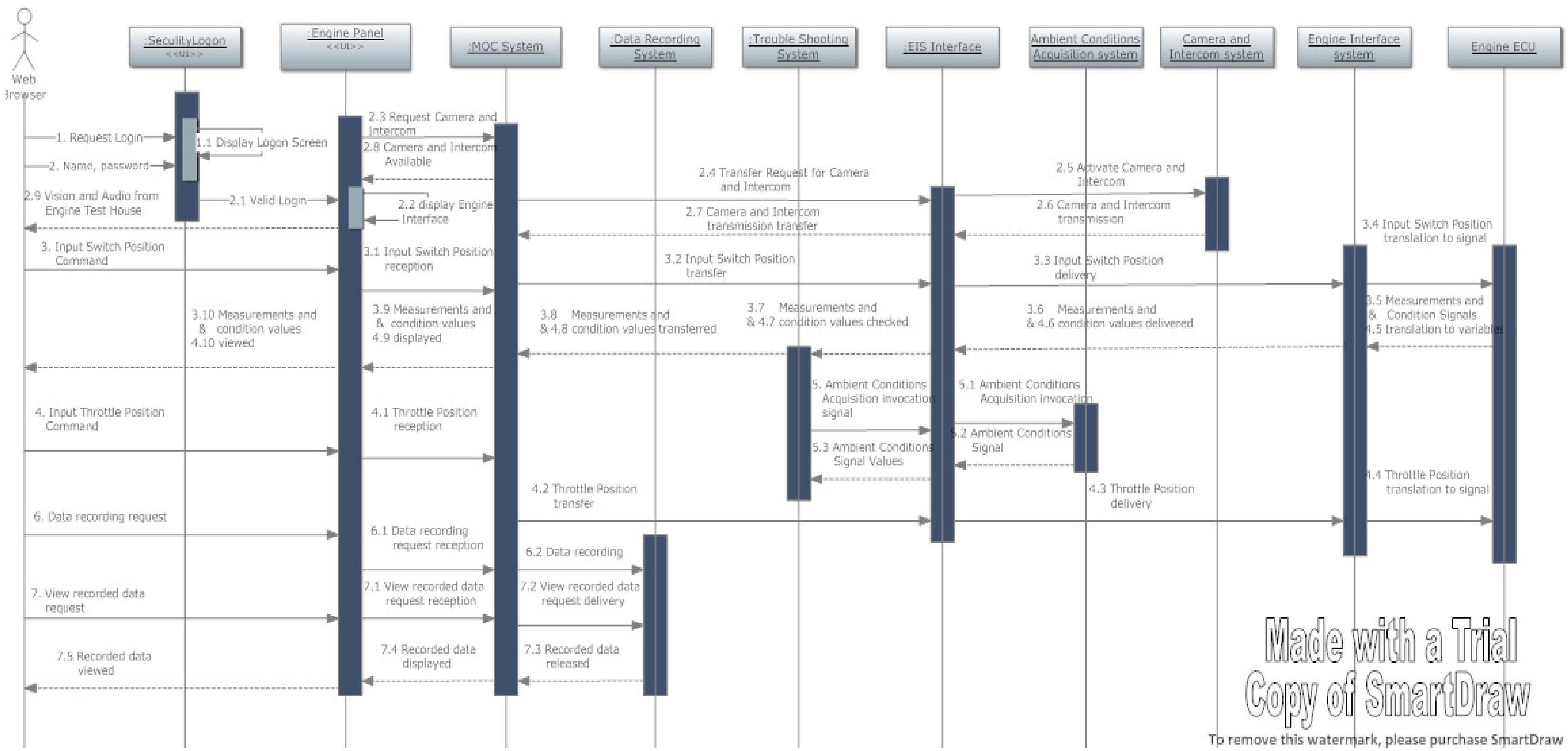


Figure 40 – Sequence diagram

4.4 General Physical Risk Assessment

Two major undesirable events with potential harmful consequences have been identified and were used as the top events for the Fault Tree Analysis (FTA) applied (Appendix B, Section B.1), based on the general architecture and components of the system in Figure 37. The first top event was considered to be the total loss of control, where the engine would run without being controlled, neither through the Internet, or manually. Individual events that may contribute to this:

A. Loss of power supply
B. Network connection loss
C. Exploit attack
D. DoS attack
E. User Misuse
F. Undetected bugs triggered
G. Manual control switch failure
H. Connection – control hardware malfunction
I. Intercom system failure
J. Camera system failure
K. Malware
L. Inside attack

The second top event was considered to be an erroneous command order incident, where the engine does not follow an intended command but instead operates at power settings other than the desired. Individual events that may contribute to this:

A. User Misuse
B. Undetected bugs triggered
C. Connection – control hardware malfunction
D. Engine Sensor Failure
E. Exploit Attack
F. DoS Attack
G. Intercom system failure
H. Camera system failure
I. Malware
J. Inside Attack

Following the FTA analysis, an ETA was conducted for both top events (Appendix B, Section B.1). A series of possible events, which if combined together would lead to dangerous or hazardous situations, was identified along with the criticality of each combination.

Moreover, a HAZOP analysis was implemented at this stage, in order to derive and explicitly state actions required for mitigation of the underlying risk. The following elements were considered for deviation from proper function, based on the architecture of the overall system:

- Junction box
- Control hardware modules system
- Measurement acquisition hardware modules system
- Control Unit
- Engine Interface Software
- Control Software
- Condition checking system
- Recorded data storage system
- Camera system
- Intercom
- Web application

4.5 Validation Plan

Validation and verification can be considered to comprise of two complementary approaches. One was the software inspections, which were static analysis of the SRS document, the design document and the source code. The other was testing which is a dynamic approach based on test data [132]. The various methods which may be applied depend on the software process model. In a waterfall model the product cannot be tested before completion whereas in incremental or evolutionary approach, testing may be applied in between stages, as in the current project. The general V&V plan follows the guidelines of IEEE-1012-2004 [72] Integrity Level 3 and DO-178B [65] for FAA Avionics Certification.

Based on Integrity Level 3 of IEEE-1012-2004, the following activities have been included in all the V&V procedures throughout the project:

- Requirements review
- Interface Analysis
- Criticality analysis
- Hazard and risk analysis
- Component test plan, procedure, design, test cases generation and execution
- Integration test plan, procedure, design, test cases generation and execution

- System test plan, procedure, design, test cases generation and execution
- Acceptance test plan, procedure, design, test cases generation and execution

According to DO-178B [69], the system was classified as of criticality Level C, based on the most severe damage expected from the implemented ETA, which was estimated that it would not exceed major damage. In this case, DO-178B requires achievement of 100% Decision Coverage and Structure Coverage of the written code.

4.6 Validation of the Requirements

The validation of the requirements has been approached by review and inspection, and by the development of a simple prototype Java SE program that simulated the various proposed components of the system and depicted their interaction.

The developed prototype (Figure 41) did not represent the actual communication modes and protocols, as it comprised of simple Java classes under the same package. The Engine Interface Software and the gas turbine were simulated with the Engine class, which received switch and throttle position input. For each throttle position value it included a predefined value of RPM, EGT, Fuel Flow, with normal distribution randomness. It returned indication to the control software for these 3 parameters. The Troubleshoot class and InputNominals() class were parts of the system that simulated the check of the engine parameters and imposed an automatic shut down if certain values were exceeded. The check referred to EGT and RPM against nominal values for each throttle position. Class Troubleshoot received nominal values from InputNominals() class, ambient conditions from Ambient class and measured parameters from the EngineDemo() class. Consequently it corrected the parameters to ISASL and checked for exceedance, in which case it conveyed Boolean variables to EngineDemo() class. Action was taken once the input values exceeded the value of 2 standard deviations of the read parameter at the current throttle setting. Finally, the EngineDemo class was the main coordinator of the interaction between the other classes and represented the Engine Interface to the user. All the aforementioned classes were sub components of the Engine Operation Control System.

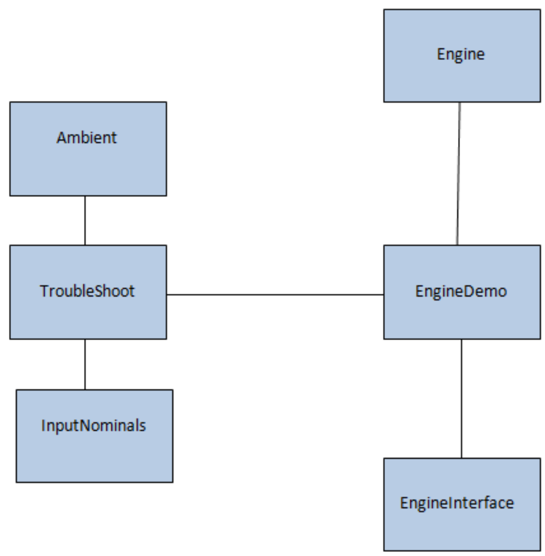


Figure 41 – Context model of the prototype program

4.7 Results – Discussion

The general risk assessment of the project revealed several vulnerabilities. These could be caused by external factors such as main power supply and network connectivity. The minimal cut sets (Appendix B, Section B.1.5) from the FTA results for the first top event (total loss of control) showed that individual event A (loss of power supply) formed a minimal cut set on its own (Equation 2), thus implying a high degree of effect on the top event. The failure of the junction switch (G) was also evident in many minimal cut sets of 2 events. Also, the different types of Internet threats (C, D, E, F, K, L) - although each one individually did not form a major invocation factor for the top event – were responsible for the formation of 8 additional cut sets likely to trigger the top event. Even the insider attack event (L) posed a threat to the system (present in 2 cut sets) and although not highly likely, it required consideration.

$$Po = A+GB+GC+GD+GE+GF+GK+GL+GH+IJB+IJC+IJD+IJE+IJF+IJK+IJL+IJH$$

Equation 2 – Boolean analysis for top event "Total loss of control"

The deployment of the ETA based on this top revealed that the worst combination may cause a major damage event.

The Boolean reduction for the second top event did not contain an individual event alone in a cut set (Equation 3). However, the large number of minimal cut sets (16) indicated a high number of combinations that could trigger the top event.

Miscommunication caused by camera or intercommunication system failure (G, H) was frequent in this series of events, along with many Internet threats.

Po = GD+GC+GA+GB+GE+GF+GI+GJ+HC+HD+HA+HB+HE+HF+HI+HJ

Equation 3 - Boolean analysis for top event "Erroneous command order"

A list of all the precautions needed to be taken was demonstrated in Appendix B, Section B.2, based on the results of HAZOP analysis for the physical risks of the project. Moreover, the elements examined in the procedural HAZOP for the SRS phase (Appendix B, Table B-2) have all been addressed and implemented. The SRS was validated by thorough inspection and discussion and the requirements have been considered feasible. The prototype program indicated that interaction between the potential software and hardware elements was also feasible.

The version of the Functional Requirements attached in Appendix A , sections A.2 - A.6, was finalised after conclusion of round 3. The requirements were not concrete from the first stage, as they would if a waterfall software process would have been followed. On the contrary, they were revisited after each round of the spiral process and modified or adapted according to findings, risk issues and alternative options applied during the phase. For example, initially, unsuccessful camera and intercom connection would not allow the program to run. This would render it very rigid – although safer – hence the alternative approach was to simply transmit live image and sound but not have it as a crucial factor, as long as there would be 2 human safety observers in the path of execution along with the program automations. Additionally, some features were suspended due to restriction of time (Data recording and retrieval) or simplified (network connection monitoring), with suggestions to be implemented at later evolution.

The objectives were all met and the procedural risks mitigated successfully. This may be noticed from the successful overall outcome of the project. If an inadequate planning from the requirements stage had taken place, it would have affected the whole work negatively. The inspection of the requirements in combination with the programmatic representation with the designed prototype, provided a safe conclusion that the project was feasible and with controllable risk, as shown by the risk analysis and the ETA. Finally, the prototype program was utilized later on in the project and formed the basis upon which the General Test Harness of the system was designed.

5 GAS TURBINE INTERFACING

5.1 Methodology

This chapter describes the methodology and the implementation of round 1 of the Spiral: the design of a program that will interact with appropriate hardware in order to interface the gas turbine. At this stage, interaction with the gas turbine was established via the ECU and the parameters available from there. Additional parameters were acquired at a later stage. The objectives of this round are listed in Table 5.

Table 5 – Objectives of round 1

Selection of Interfacing Hardware
Selection of Interfacing Programming language
Design of appropriate programs to interact with hardware and gas turbine
Physical connection of gas turbine and interfacing hardware
Determine Feasibility of next round
Adequate validation and testing

The Engine Interface Software (EIS) phase involves the translation of software commands to hardware signals to the engine and vice versa. MATLAB and LabVIEW programming environments were considered. The latter was selected because it offers a great variety of functions that ease data acquisition and signal generation with directly compatible hardware components, whereas MATLAB would require a special toolkit. LabVIEW 2011 SP1 for Windows 7 was used. LabVIEW is a drop – and drag design language based on programming language G, developed by National Instruments [100]. It includes a large variety of functions that integrate base methods in order to provide convenient tools for a number of functionalities, such as signal generation (analogue or digital), data acquisition, embedded control, control simulation. It also has multitasking capabilities, allowing for parallel execution of the various functions.

Two analogue signals were generated for the engine ECU: One ranging from 0 to 5 Volts DC for the START/STOP 3 position switch and another from 0 to 5 Volts for the throttle knob. The readings of EGT, RPM, the status of the engine and the ECU error messages are transmitted in the form of serial data (RS-232). The hardware connected to the engine was the NI 9174 compactDAQ unit [100] and the C series NI 9263 analogue output (-10 to +10 Volts) module [101]. The serial data is driven into the PC through a serial to USB converter.

The physical Risk Assessment was accomplished with the use of FMEA and FMECA [19]. This method can be used and may be applied at this stage, as the system is defined enough to be presented as a functional block diagram. FMECA also accounts for the criticality assessment of the design. HAZOP [47] was applied for the procedural risk assessment. The reason for this selection was that the analytical layout of HAZOP could provide the opportunity to address the mitigation actions directly, even for the procedural risks, in a similar manner when applied in physical risk analysis.

Validation included unit testing of individual functions, inspection of the code and control flow testing of the complete functions. Finally, integration testing was accomplished after integration of the functions by reapplying the test cases previously identified for the control flow. Control flow testing was based on the derived control flow diagrams of the designed software components, which were used to define the appropriate test cases. The cyclomatic complexity suggested by Tom McCabe [147] was calculated and used as a guide to identify the independent paths of the designed methods, in order to create the required test cases.

5.2 Risk Assessment

The process has been examined for underlying procedural and physical risks. The physical risks were approached by FMEA and FMECA. The NI hardware components have known MTBF. For the electric and electronic components that do not have published MTBF, once they have been purchased from reliable manufacturers and comply with the Mil– HDBK - 217F standard, it has been assumed that they have similar MTBF standards. Any part that has been modified in the lab to match specific needs of the project has been assumed to have one level of failure frequency higher (Appendix B, Section B.3).

The highest scoring physical risks reached the value of 9 (undesirable) according to FMECA and they involved looseness of cable connections and power supply to the system. For these reasons, special care has been taken to secure the wiring and the routing of the cables. Also, the system is protected against power loss or surge with a power stabiliser and at a later stage, the mains power supply monitoring was implemented. One of the elements of the procedural risk assessment was – amongst others - the consideration of cost. Hence in order to maintain the cost of the project low, investigation for reusable existing components (hardware or software) was conducted and a wide search of the market took place (Appendix B, Table B-10).

5.3 Analysis, Design and Implementation

5.3.1 Software

The EIS was designed as a LabVIEW VI with 3 main functions. The function ECUSerial receives the serial data transmitted from the ECU: RPM, EGT, throttle position, fuel pump out voltage, ECU supply voltage, error conditions of the ECU, and the status of the engine. The data acquired from the ECU were categorised as the Normal data set, the Alternate data set and the Error data set. Data are transmitted in blocks of 6 bytes comprising 8 bits each at a rate of 48 bytes per second. During normal operation the normal data set is continuously transmitted and every 3 seconds the Alternate data set is transmitted. Error data set is transmitted only when an error occurs. The value of the last byte of the block determines which set type will follow. The bytes are transmitted as unsigned 8 bit characters. LabVIEW reads them in as ASCII characters.

The attributes of this function were the characteristics of the serial data transmission of the ECU (Figure 42). These were entered into the VISAConfigure sub function. The sub functions that follow read in the data as string characters (ASCII), cast them to 1 – dimensional arrays of unsigned characters and then index the individual elements according to the manufacturer's serial transmission protocol. After appropriate manipulations the data are presented on the front panel of the VI. The operation programmatically stops if an error in the data input occurs. Thorough description is shown in Appendix E, Section E.1.3.

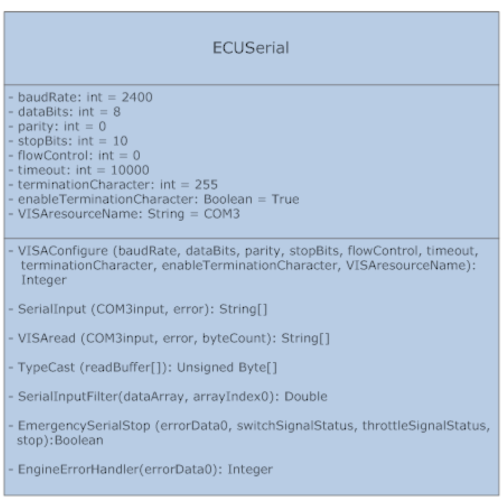


Figure 42 – UML Class diagram of ECUSerial function

The attributes of the SwitchSignal function (Figure 43) for the start/stop switch signal generation were: the name of the physical output channel, the minimum voltage value, the maximum voltage value and the measurement unit of voltage. These were entered into the CreateVirtualChannelSwitch sub function which configures the signal at the indicated channel. The output value can be controlled by the user through a slider control in the SwitchVoltage sub function, where every change imposed by the user from the front panel is implemented through the SwitchWrite sub function. The sub VI ceases to operate if an error occurs, the user presses the Emergency Stop Button, or the serial input from the ECU is lost. A detailed description of the function is shown in Appendix E, Section E.1.1.

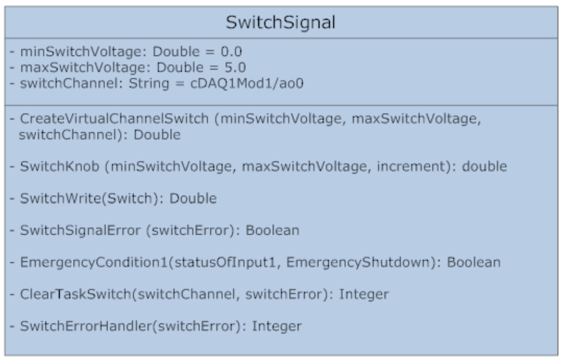


Figure 43 – UML Class diagram of function SwitchSignal

The ThrottleSignal function (Figure 44) for the throttle signal generation was designed almost identically with the SwitchSignal function. The attributes were entered into the CreateVirtualChannelThrottle sub function, which configures the signal at the indicated channel. The output value can be controlled by the user through a slider control in the ThrottleSlider sub function, where every change imposed by the user from the front panel is implemented through the ThrottleWrite sub function. The function ceases to operate if an error occurs, the user presses the Emergency Stop Button, or the serial input from the ECU is lost. A detailed description is shown in Appendix E, Section E.1.2.

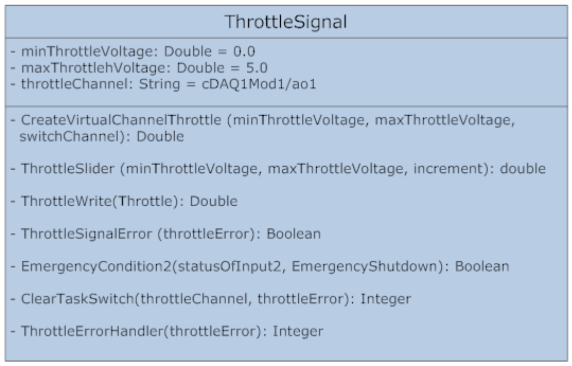


Figure 44 - UML Class diagram of function ThrottleSignal

The calibration between the throttle and the signal was based on the relationship observed between the existing manual throttle knob and the throttle feedback indicated on the Engine Data Terminal (EDT). The full active route of the knob was measured 225°. Divided in 100 intervals of 2.25°, the response of the throttle indication was observed for each incremental movement of the knob by 2.25°. The relationship was found approximately linear, with a low level of non linearity for low values of the knob setting that diminished after around 40% setting (Figure 45). Therefore, the relationship between the throttle knob and the generated signal was able to be considered linear with a very good approximation.

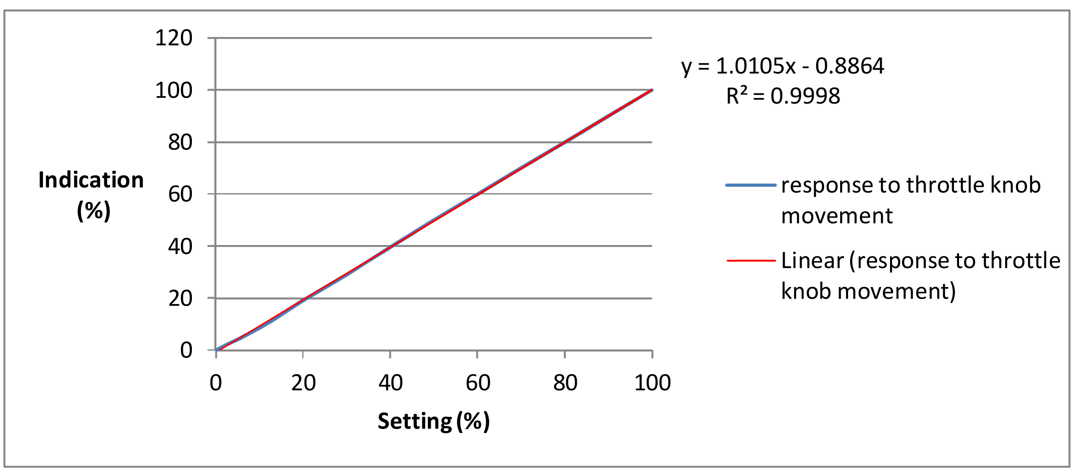


Figure 45 – Relationship between the manual throttle knob and the throttle indication

Due to the linearity between throttle knob and indication, the implemented throttle in the software was also assumed to have a linear relationship between the movement and the throttle signal generation. The maximum value of the generated signal was increased until the throttle indication on the EDT reached 100%. The signal value was observed at 4.652 Volts. Hence, with 0 Volts corresponding to 0% throttle setting and 4.652 Volts to 100%, the slider was configured to increase the signal value in increments of 0.0465 Volts, for each one of the 100 intervals of the full route (Figure 46Figure 1).

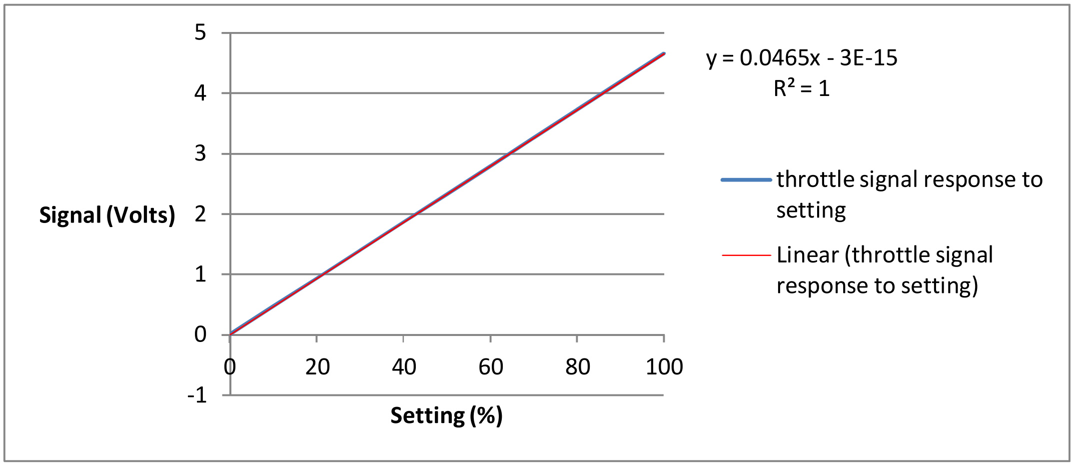


Figure 46 - Relationship between the software throttle slider and the throttle indication

All the aforementioned designed functions were included in a single VI (EIS.vi) and operate in parallel mode when they are invoked. They interact when:

- The user presses the Emergency Stop Button. Then the throttle signal is set to 0% and after 3 seconds delay the Switch signal VI is set to AUTO STOP position, allowing the engine to stop smoothly. The ECUSerial function continues to operate, even when the engine has been shut down.
- An error in ECUSerial occurs. Then the other 2 functions follow the previous shutdown sequence.
- An error in the signal generation functions occurs. Then the 'TAKE MANUAL CONTROL' indicator is invoked and illuminated.

5.3.2 Hardware

Channel C (switch channel) and channel D (throttle channel) of the ECU were branched by connection of Y – lead splitters, contained in the configured junction box. Each extension was then connected to a switch that disconnects the input signal. For each splitter installation, only one switch may be closed every time, allowing signal input either from the PC or from the manual control box. Simultaneous signals counterbalance the voltage hence there is no input to the ECU channels. One branch of each extension was routed to the manual control box of the engine. The other branch was routed to the NI 9263 module, ports 0 and 2 for the positive (+) wires, and ports 1 and 3 for the neutral. The module was installed in slot 1 of cDAQ – 9174 data acquisition unit which in turn was connected to the PC via USB (Figure 47).

Similarly, the Telemetry output of the ECU was branched in the junction box and one branch was routed to the existing Engine Data Terminal (EDT) and the other to the PC port COM3, via a serial to USB converter cable. No switch intervenes in this circuit, allowing the monitoring of the ECU output both at the EDT and the PC simultaneously (Figure 47).

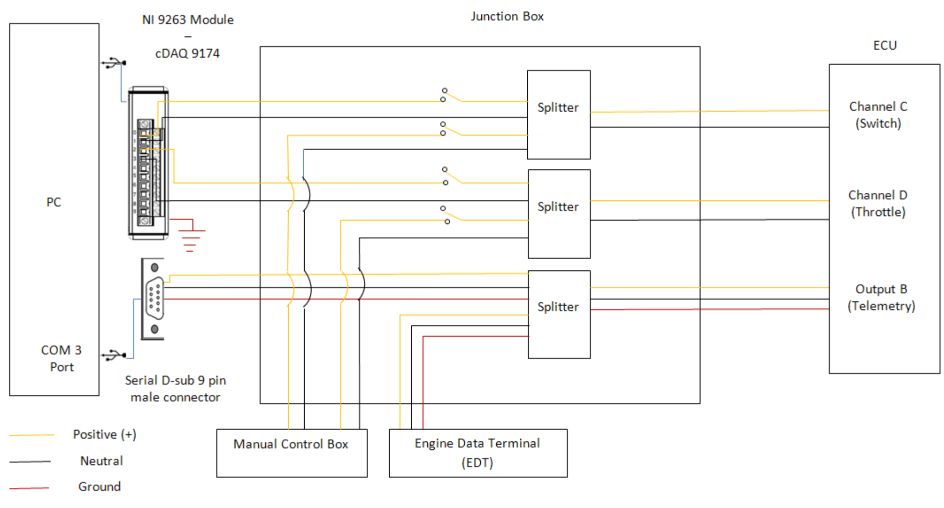


Figure 47 – Wiring diagram of the hardware connection to the ECU

5.4 Validation

SwitchSignal and ThrottleSignal functions validation was accomplished as shown:

- LabVIEW Unit Test Framework (UTF). Values were imported from the front panel of the sub VIs and checked for correct expected values at the designated indicators.
- Signal measurement. The analogue DC output was physically measured with an oscilloscope.
- ECU indications in LabVIEW were compared to the EDT indications simultaneously, when the engine battery was in ON position.

ECUSerial individual validation:

- Unit testing by implementation of serial loopback test. The function VI included a Write sub function which, when enabled, allowed the user to enter data in the form transmitted from the ECU and check whether they were read as expected. The transmission and reception was achieved by connecting pins 2 and 3 of the serial port of the computer.
- ECU indications in LabVIEW were compared to the EDT indications simultaneously, when the engine battery was in ON position.

- ECU indications in LabVIEW were compared to the EDT indications simultaneously, with the engine.

The validation of messages that are not transmitted under normal operation conditions was achieved by simulation with the loopback approach. Consequently, control flow testing was accomplished for the 3 designed functions (Appendix C, Section C.1).

Integration testing of the 3 functions into EIS.vi reapplied the previous testing procedures:

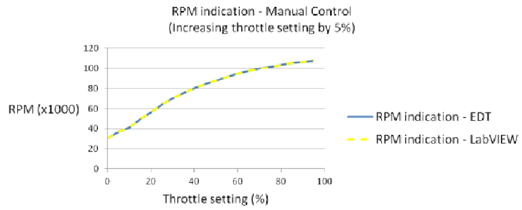
- ECU indications in LabVIEW were compared to the EDT indications simultaneously, with the engine.
- Serial Loopback test was applied to simulate all the cases where error messages are transmitted, as this was not possible during normal operation.

Control flow testing of the integration is shown in Appendix C, Section C.1.4.

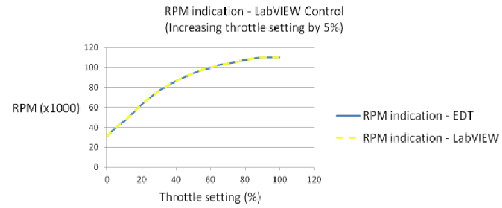
5.5 Results – Discussion

The individual function testing has produced successful 100% statement and decision coverage. The integration testing which took place at fully operational conditions of the engine produced 100% statement and decision coverage of the EIS as well. Function ECUSerial was inevitably complex with a cyclomatic complexity VG = 20, which was much higher than the conventional limit of 10. The reason was that it had to encounter every different ECU response and create a corresponding message. Albeit this complexity, it has been possible to derive the required test cases.

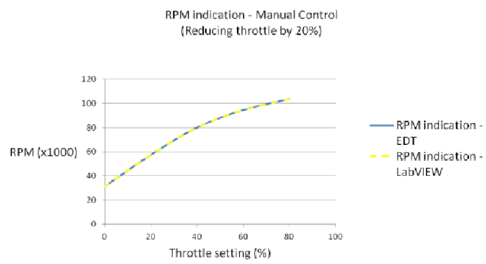
The throttle indication was tested both from the LabVIEW interface and the manual throttle knob. The settings were changed from 0% to 100%, first from the interface and then from the knob. The feedback indication from the ECU was observed from both, the LabVIEW interface and from the EDT simultaneously. The results showed that the interface slider incremental setting variation had totally matched the knob incremental setting variation. Consequently, the engine was initially started from LabVIEW and operated all the way up to 100% throttle by increments of 5%. Then the throttle was reduced by 20% at each step down to 0% (idle), increased again to 100% by 20% and reduced in one step to 0%. Transition to manual operation control followed and all the previous sequence was repeated. Finally, transition to PC control was re-established and the engine was shut down by pressing the Emergency Shutdown button. During the previous manipulations, the readings of LabVIEW and EDT were monitored and the results were plotted in graphs for comparison (Figure 48, Figure 49).



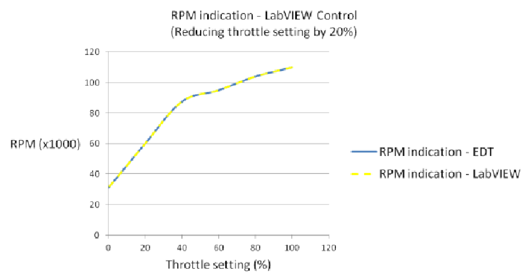
(a)



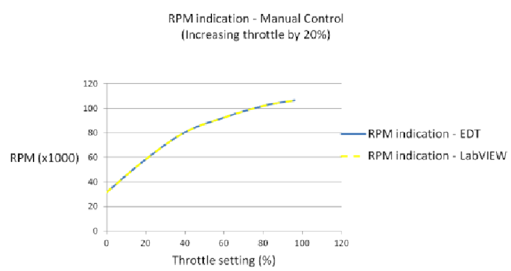
(b)



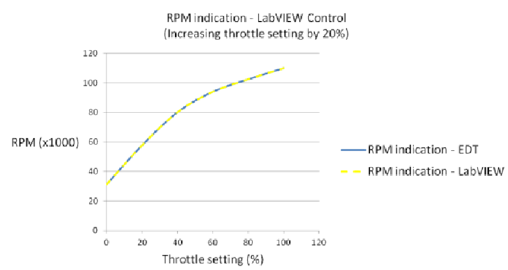
(c)



(d)

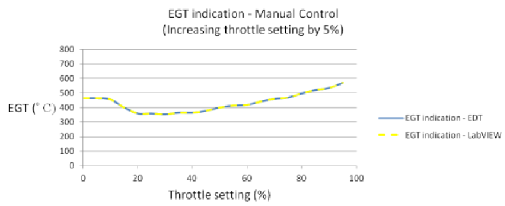


(e)

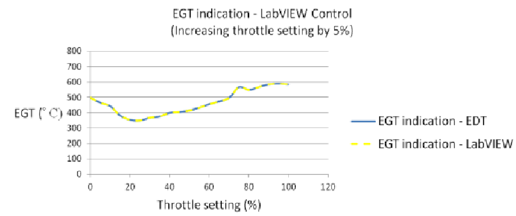


(f)

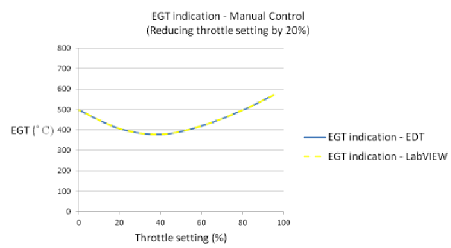
Figure 48 – RPM indication



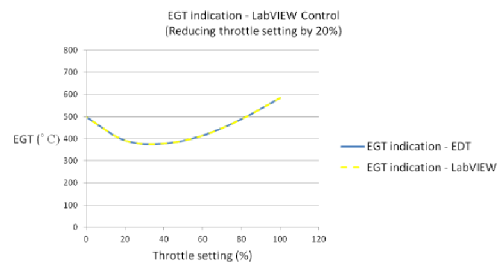
(a)



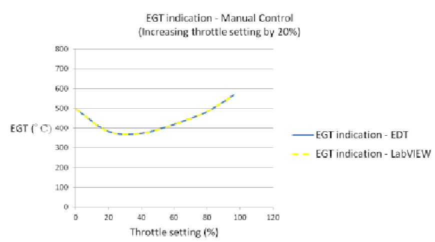
(b)



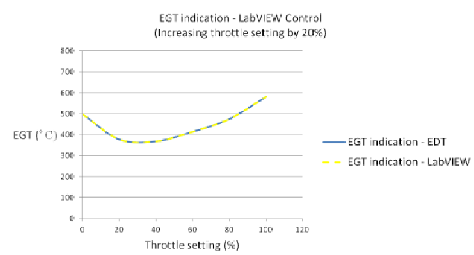
(c)



(d)



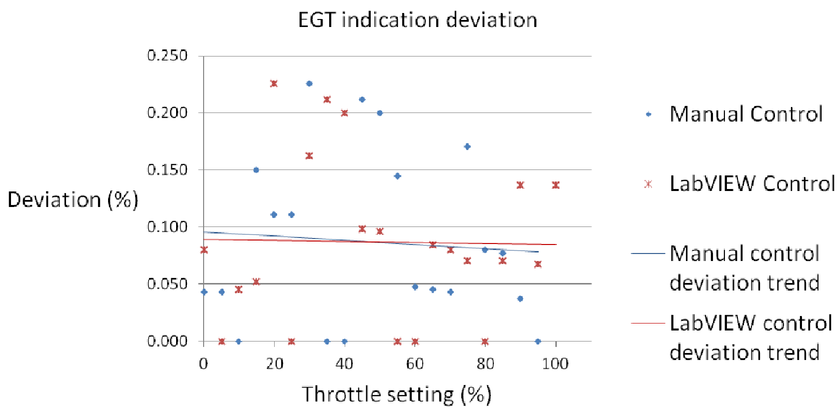
(e)



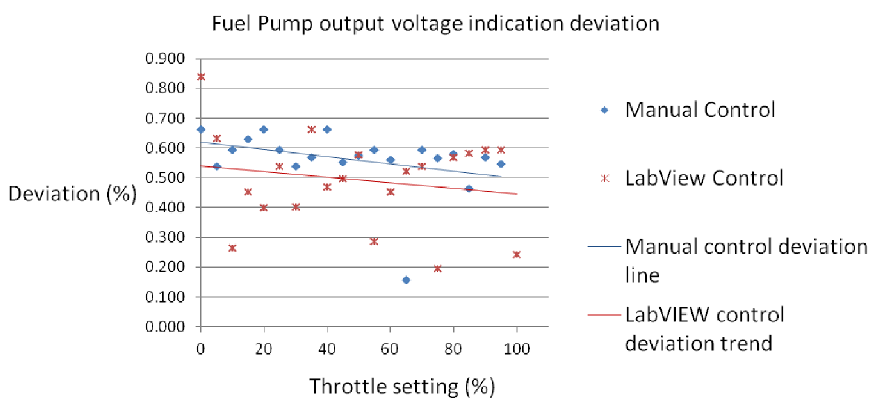
(f)

Figure 49 – EGT indication

The results showed 100% match of the RPM indication. The EGT was nearly 100% matched (deviation below 1%) and the only deviation between the EDT and LabVIEW was due to the representation difference, as on LabVIEW EGT was depicted with double precision whilst on the EDT as an integer (Figure 50.a). Negligible deviation (below 1%) was also observed at the fuel pump output voltage, where the difference occurred in second and third decimal digits, again due to the numerical representation and different accuracy settings between LabVIEW and the EDT (Figure 50.b). The throttle indication was 100% matched and also the ECU supply voltage was identical. The indications in LabVIEW responded rapidly to the throttle manipulation, either manually or from LabVIEW without any noticeable latency between the EDT and the PC.



(a)



(b)

Figure 50 - Deviation between indications of LabVIEW and EDT during manual and LabVIEW operation

It was also observed that the throttle reached 100% setting at around 4.7 Volts DC signal, rather than 5.0 Volts as expected. However, as long as the settings generated from LabVIEW match the settings generated from the manual control box, this deviation was considered acceptable. Besides, it had been mentioned by the Manufacturer that the signal values were approximate. This was attributed to the nature of analogue signal, whereas with digital signal values signal generation would be precise. Finally, at the day of the final validation test of the EIS, the manual throttle regulator was able to reach only 96% throttle setting. This fact did not affect the overall test.

A function not included in the design was the remote operation of the battery switch of the engine. This was a simple 2 position switch that can easily be replaced at any time with a relay switch, accepting a signal from the EIS. The EIS will need to be modified to include an additional signal generation function. However, to simulate this operation, a battery toggle switch was placed in the operation interface, representing a Boolean parameter in the program, which if not selected it will not allow the other functions of the system will not operate. This feature has been carried along the next phases of the program as well.

As a conclusion, at this point an independent application for a standalone computer was completed, with an ergonomic operation panel. Controls were placed left, indications in the centre of the panel, with EGT and RPM most noticeable, and warnings and status on the right Figure 51.

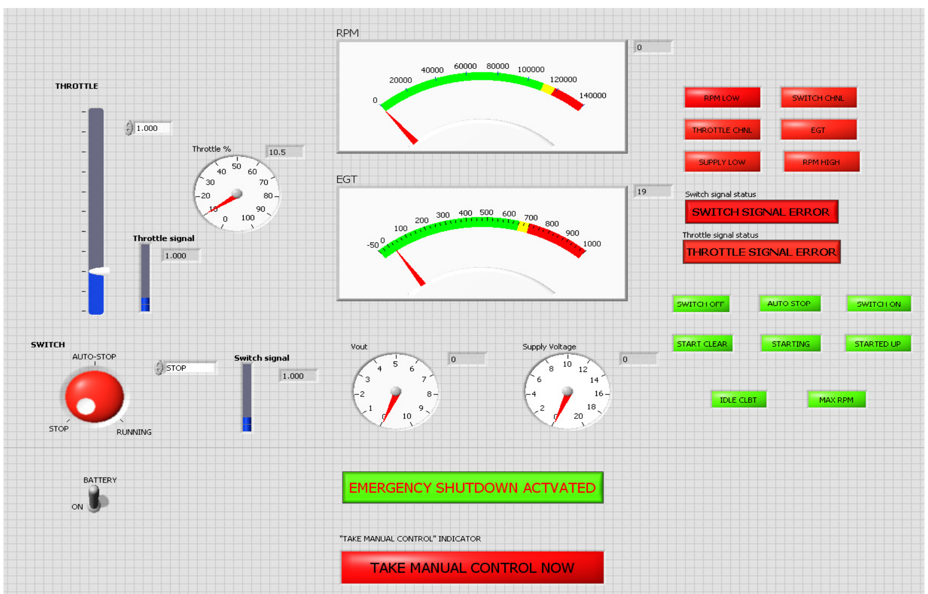


Figure 51 – Stand alone application Operation Panel

6 ENGINE OPERATION CONTROL SOFTWARE

This chapter describes the methodology and implementation of round 2 of the Spiral process. It discusses about the core program – Engine Operation Control Software (EOCS) design and interaction with the Engine Interface Software (EIS). By the end of this phase, a basic LAN version was able to run but yet not completed and safe as required.

6.1 Methodology

The EOCS was broken down to 2 basic software sub components: the EIS Interface (EISI) and the Main Engine Operation Control Software (MEOCS). The objectives of this round are listed in Table 6

Table 6 – Objectives of round 2

Design other language program and invoke EIS
Establish connection between EIS and designed program on the same computer Exchange data between EIS and local program
Determine position of the Main Engine Operation Control Software (MEOCS)
Design the MEOCS
Establish connection between local program and MEOCS and exchange data
Design integration testing harness

6.1.1 Approach to the software modules design

The EOCS modules were written in Java SE, JDK 7u17, despite the security issue that had occurred in 2012 for JDK versions earlier than 7u10. The reasons for this selection were: Java has great capabilities to design graphical user interfaces, multitasking applications and very efficient procedural functions within LAN [114]. Another very powerful feature of Java is garbage collection. This is an automatic memory management process, where memory used by objects is de-allocated by the program when they are no longer needed. Java determines implicitly if an object is no longer needed when it has no more live references to it [23], [118]. Garbage collection prevents continuously increasing memory consumption - known as memory leak - where objects are constantly stored but cannot be accessed. This feature is a great advantage over C++, although that C++ would communicate easier with native language programs in any operating system.

Primarily, the EISI needed to invoke the EIS thus a Java program invoking a LabVIEW program. Building a DLL or an .exe file was not an important difference in LabVIEW but their invocation was different. Once a DLL file is loaded, the program will hang until completion of the DLL program. To overcome this obstacle, the DLL must be loaded on a dedicated thread (multitasking). The other option was execution of a Process object in Java. By implementing the execution of a Process, the Java program does not hang until the execution of the child program is completed. Instead, it continues executing the subsequent statements. However, the child program (.exe file) must be shut down explicitly, unlike the DLL approach. In this project, invocation of the EIS takes place by executing a LabVIEW .exe file as a Process object. When the parent program is closed, the child program will be closed explicitly.

After invocation, it is essential to establish an interface to connect the EISI and the EIS for bidirectional handling of data. The TCP (Transmission Control Protocol) protocol of the Transport layer acts as the main mean for transmission under HTTP and FTP. However, UDP (User Datagram Protocol) may be preferred in some cases instead of TCP. TCP attempts to provide a reliable connection [74] and requires acknowledgment between sender and recipient. The shortcoming of this protocol is that it implements slower communications compared to UDP [74]. On the contrary, UDP does not apply acknowledgement but requires less overhead and can implement faster communications compared to TCP. It is used by application layer protocols such as RTP which transmits and receives real time data, including voice or image [74]. Data order and quality is not guaranteed upon delivery and require additional manual implementation from the developer. TCP was selected in order to reduce additional catering to ensure the exchanged data quality.

One of the objectives of this round was the determination of the position of the MEOCS software module. EISI, EIS and ACAS need to be installed on the computer directly connected the gas turbine and the hardware. The MEOCS can be installed either on the same computer or on a remote one. Local installation may result in a very high CPU usage of the host computer, as it already requires considerable memory consumption to run the LabVIEW components. Separate installation of the MEOCS would imply the existence of inevitable latency present at the physical layer, which comprises of the wiring within the network and the interacting computers, as discussed in section 3.8.3.

Wherever GUI's are instantiated, the Singleton design pattern according to Gamma et. al. [49] was applied. The Factory design pattern was applied for the configuration of the engine data to be transmitted to the next tiers. For this to be accomplished, the data configuration follows the guidelines of a Java interface, the concrete implementation of which will vary for different gas turbines.

The program was also investigated for the latency of the command to reach the EISI, after reception from the user interface end, with the conduction of a simple comparative experiment [97]. Time was measured by implementing the nanoTime() function of Java System object, which returns the current value of the most precise available system timer, in nanoseconds [115]. Elapsed time until the response from the EISI was measured, and it was assumed that the time for the command to arrive at EISI was half of the total elapsed time. Box plots were designed to depict the results [97].

6.1.2 General test harness

As indicated by the last objective of the current round, in parallel with the implementation of the core software modules, the General Test Harness (GTH) was designed, in order to be used for integration testing of the components. This was also built with Java SE JDK 7u17. This program simulates the operation of the EIS and the gas turbine integrated. It is invoked identically and it implements the same protocols of data transmission to the EOCS. The nominal values of the engine parameters were established after experimental sampling with the gas turbine running throughout the whole throttle range. RPM and EGT were corrected to ISASL conditions and the standard variation of the parameters were determined with the analysis of the variance of a single factor, as described by Montgomery [97] (throttle setting was considered to be the single factor). Only 13 full throttle cycles were conducted, as the gas turbine had a short time interval remaining until overhaul.

6.1.3 Validation methodology

Validation included unit testing of individual functions with JUnit [52], inspection of the code and control flow testing of complete functions. Integration testing was accomplished after integration of the individual functions and the use of the GTH. Control flow testing was applied during the integration testing, based on the derived control flow diagrams of the designed software components and the cyclomatic complexity suggested by Tom McCabe [147], in order to identify independent paths of execution.

6.2 Risk Assessment

Risk assessment of this stage was accomplished with HAZOP. Only procedural risks were considered, as this round has not introduced anything that would imply the necessity for further physical risk assessment (Table B-11).

The risk assessment addressed amongst others, the clarity of the objectives, identification of software dependencies, and proper installation of the components. These observations directed to the interfaces between the currently introduced components (MEOCS and EISI) and the EIS, and also the location of the new components. The latter had to be considered, in order to optimise performance and flexibility. However, for a LAN application, the two components should be separately installed. On the contrary, for a web version, they could be either located together or separately. Hence the optimum solution was to design an interaction of MEOCS and EISI with Java RMI, allowing provisions for both options – remote or local installation - without the necessity for any modifications.

6.3 Analysis, Design and Implementation

Description of the design and implementation of this round was considered in four sections: one for the EIS adaptation, one for each module of the EOCS and one for the test harness (Figure 52).

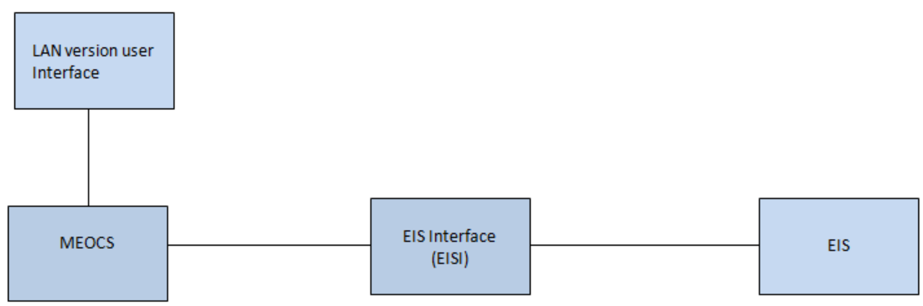


Figure 52 – Context model of EOCS and correlation with EIS

6.3.1 EIS adaptation

The EIS was configured with a TCP server to accept commands and a TCP socket to send the engine output data to the EOCS. Additionally, a new function was added to notify the remote user about the status of the system. This included the following messages:

- SYSTEM OK
- SYSTEM NOT READY
- NO SERIAL INPUT
- SWITCH SIGNAL ERROR
- THROTTLE SIGNAL ERROR
- SWITCH SIGNAL ERROR; SERIAL INPUT ERROR

- SWITCH SIGNAL ERROR; THROTTLE INPUT ERROR
- SWITCH SIGNAL ERROR; THROTTLE SIGNAL ERROR; SERIAL INPUT ERROR

The final output of the EISI towards EOCS was configured prior to transmission. The data were grouped into 4 main categories: system status, engine status, engine warnings and engine indications. They leave the EIS after a trivial configuration into double numbers of 6 digits, placed into specific order. They are then converted into a String variable from LabVIEW and sent to Java. Standardised configuration takes place in EOCS. As for the commands, they end up in the EIS as a String that is cast into an array of double variables. The first 6 digits of the string represent a flag number indicating to which control the command corresponds. The last 6 digits represent the new value of the desired control.

6.3.2 EIS Interface

This is a two package Java program interacting with the EIS. Upon start it instantiates an RMIServer() class object. This is the RMI server awaiting connection. All the other objects are instantiated after connection with the RMI client. The program was designed for multithread operation, with one thread dedicated to commands transmission (class ClientEIS) and another to engine data reception (class ServerEIS). The containing classes override the run() method of interface Runnable() and include appropriate TCP sockets and server sockets to communicate the data (Figure 53).

Once the engine data are received from the EISI, they are checked for correctness. If a corrupted set of data is received, it is replaced by a default correction value momentarily, preventing any Input/Output exceptions to occur. Consequently, via TCP connection sockets, they are transmitted to the next tier, which is the MEOCS (Main Engine Operation Control Software), after further configuration, as indicated by the MessageConfiguration() interface. The implementation of the configuration takes place in the class MessageImplementation(), where each group of information is flagged appropriately and so are the engine indications individually (Figure 54). As for the command input, after being received from the buffered reader of the EISI, it is checked for correctness and then sent towards EIS in the finalized form (Figure 53).

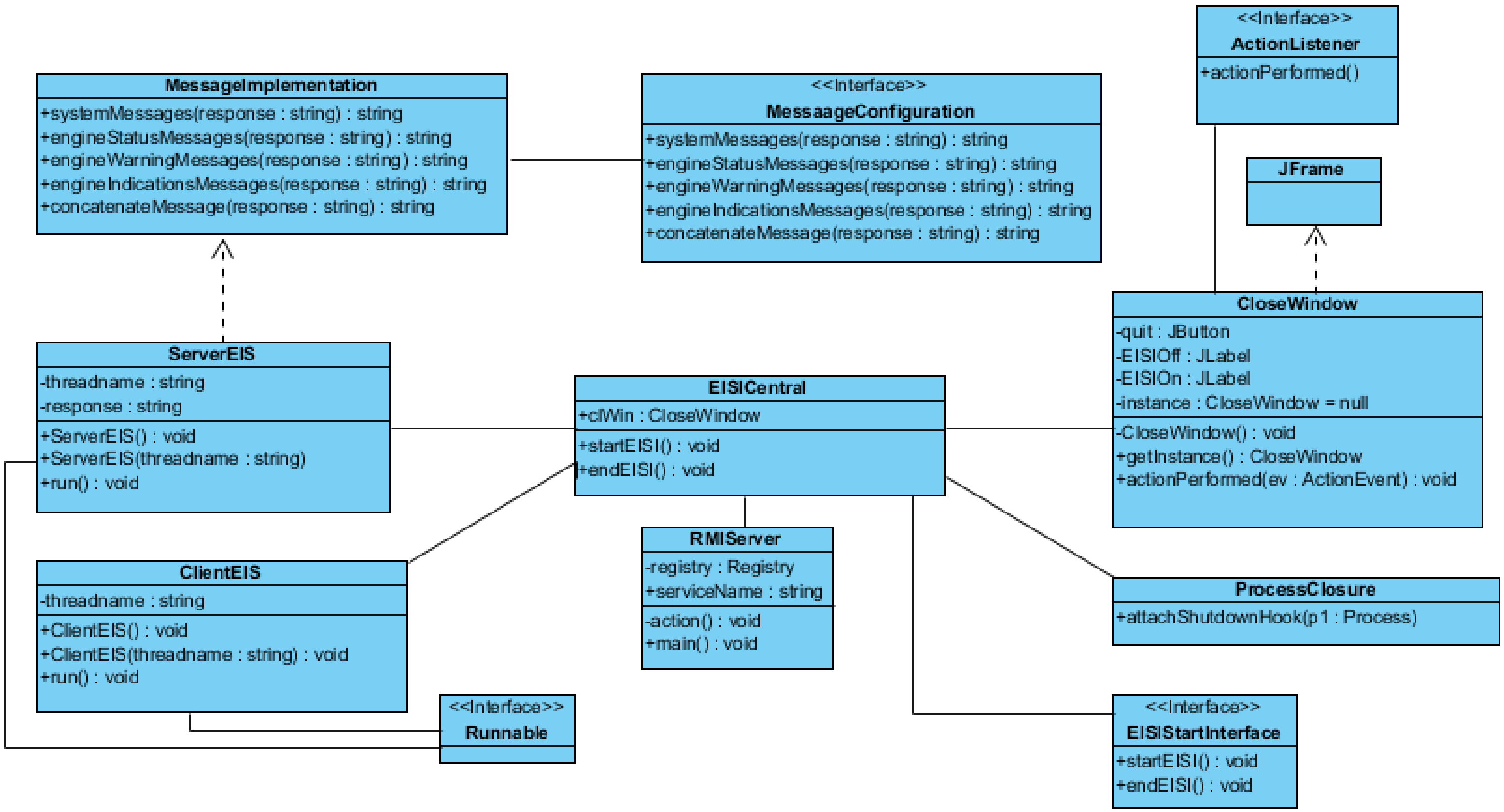


Figure 53 – UML Class diagram of EISI

@ end;NORMAL; end;SYSTEM OK; end;AUTOSTOP;START CLEARANCE; end;_; end N01.00.0 T06.015.0 Throt0.0 Voutp0.0 Vsupp0.0 Fflow0.0 end $

Figure 54 – Gas turbine output message format

6.3.3 MEOCS

This module has been designed including an RMI client in order to bind to the RMI server which consequently invokes the EISI (Figure 55).

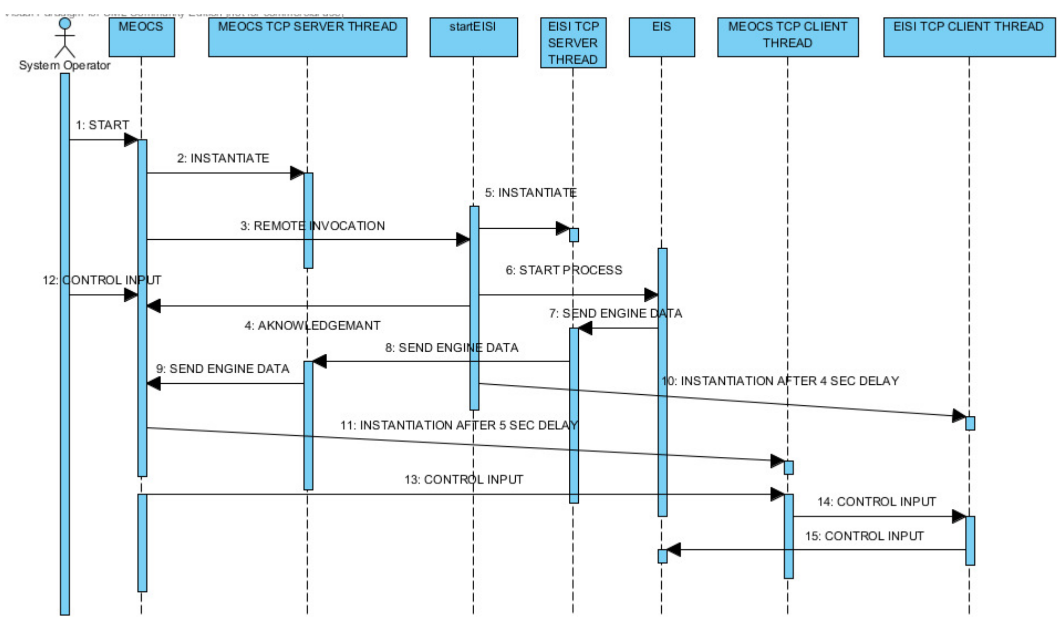


Figure 55 – System RMI invocation sequence

When the MEOCS client is disconnected, the EISI shuts down. Thus Java Shutdown hooks have been attached in the EISI so that upon closure to destroy the child process of the EIS, preventing the gas turbine from running unattended. The RMI clients and servers on each side are closed and the connection is released and available for the next user. The RMI server is restarted form an independent program, which monitors the operation through a handshaking TCP connection, in a new JVM instance. Also, a very simple LabVIEW VI (SignalClosure.vi) was designed on the EIS side, which is invoked by a shutdown hook upon closure, to reset the signals sent to the engine: throttle to 0% position and switch to auto stop position after 3 seconds delay. This feature will always shut down the engine safely after the remote user quits the client abruptly (Figure 56).

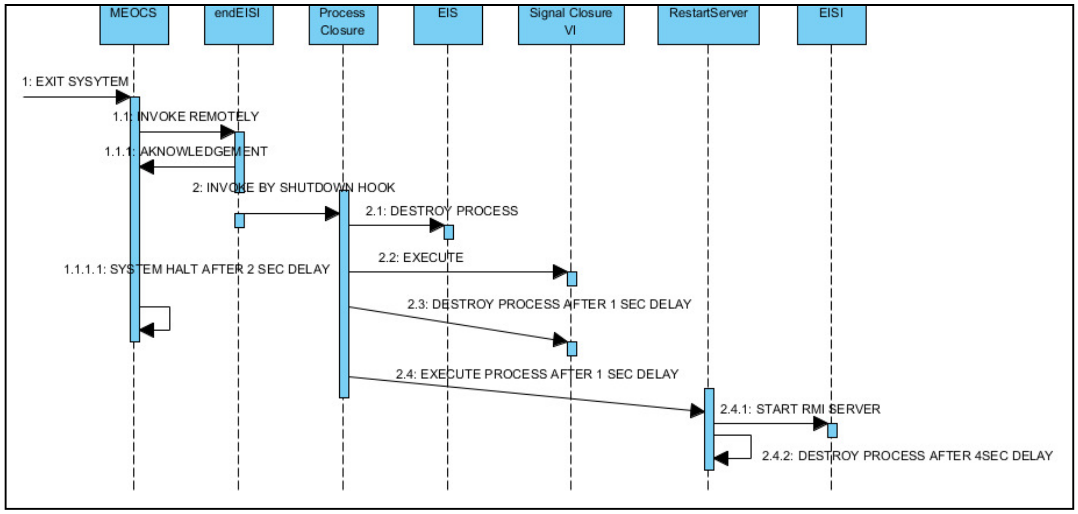


Figure 56 - System closure sequence

At this stage, the MEOCS was designed as client with a Java GUI to represent the engine operation panel. The class that instantiates the GUI has been designed to adhere to the Singleton pattern, preventing multiple instantiation. That was accomplished by implementation of a private constructor and instantiation of an instance of the class object only if it is null (i.e. the GUI has not been previously instantiated) (Appendix E, Section E.4.15). It was designed for multithread configuration, with a dedicated thread for engine data reception (class MessageReceiver) and another for command transmission (class CommandSender). These classes both implement the Runnable Interface (Figure 57).

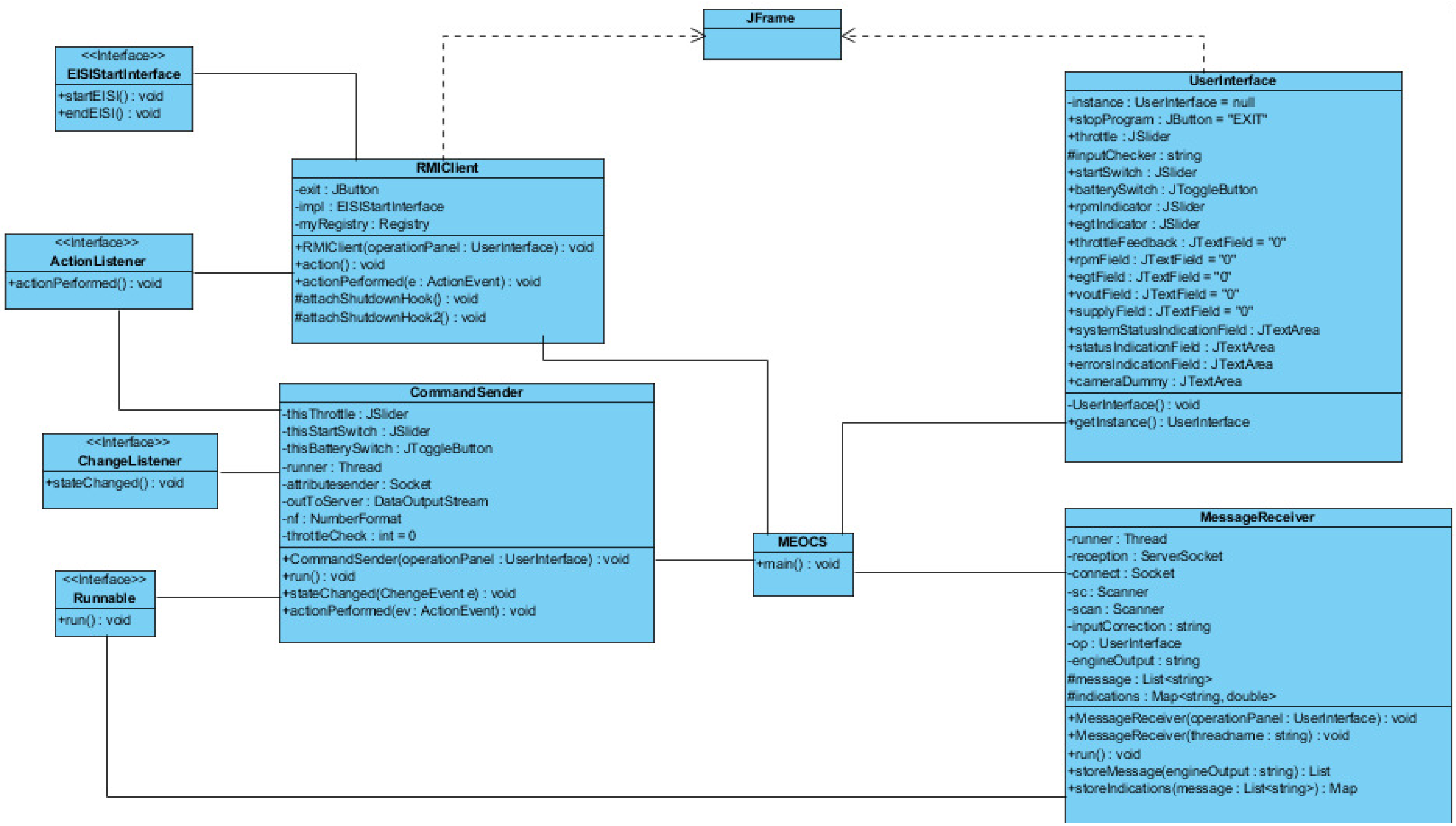


Figure 57 – UML Class diagram of MEOCS

The engine data output upon reception are configured in Java list arrays and maps, after being identified from their accompanying flags. That is how the GUI recognises them and assigns the appropriate values to the correct fields. The operation panel has been designed ergonomically and with clarity to prevent user confusion and misuse. The controls were situated on the left side of the panel, the main indications on the centre and the secondary engine indications along with ambient conditions were located on the right side. The caution panel was placed on the top of the panel whilst the emergency exit button on the bottom (Figure 58).

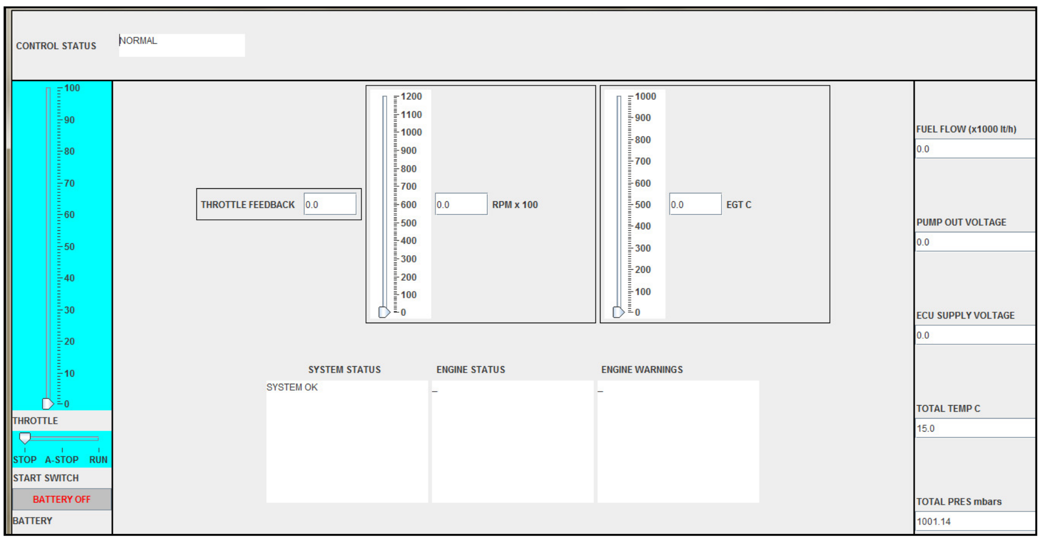


Figure 58 – MEOCS GUI

It was very important to ensure the solidarity of the operator during the engine run. To achieve that, an additional handshaking TCP connection was introduced between the MEOCS client and the EISI RMI server. The connection is closed right after the RMI binding and reopened only after closure, when the RMI server has been reset. Hence, when the GUI is loaded, no other user can access it.

6.3.4 The General Test Harness

This was developed primarily as a tool for integration testing of the various modules and also for the final validation tests of the final system. It was not intended to simulate the thermodynamic processes of the engine and simulate its performance. Thus, the data included were derived from experimental runs, as described earlier in this chapter. Also, the architectural design was not of such importance for this program, although the Singleton pattern was also applied to prevent multiple instantiation of the GUI. The layout was simple and it was based on the prototype program developed in round 0.

It maintained the same communication protocols as the EIS and transmits the engine output data using the configuration established in the EIS as well. It was designed to perform multithread operations, with one thread running the GUI and accepting the command input from the EISI, and the other to transmit the engine data back to the EISI, always according to the switch and throttle settings. It transmits constantly, similarly to the transmission rate of the engine and the EIS integration (Figure 59).

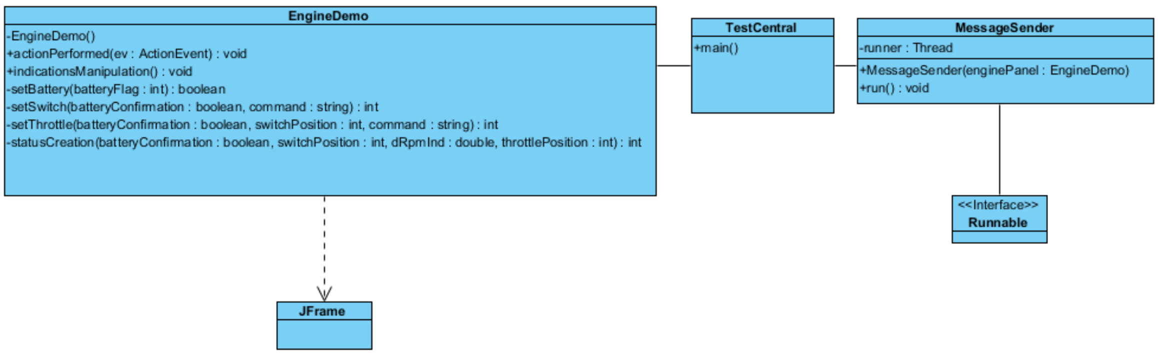


Figure 59 – UML Class diagram of the General Test Harness

The functionality was configured to allow for the following:

- Implant system status errors
- Implant engine ECU errors
- Implant RPM and/or EGT exceedance
- Implant mains power supply loss

All the above conditions can be cleared by the user and were considered to match all the possible outcomes from the original engine and the EIS. The indication fields of the harness include the actual EDT indication fields and all the messages that were also added to the MEOCS GUI (Figure 60).

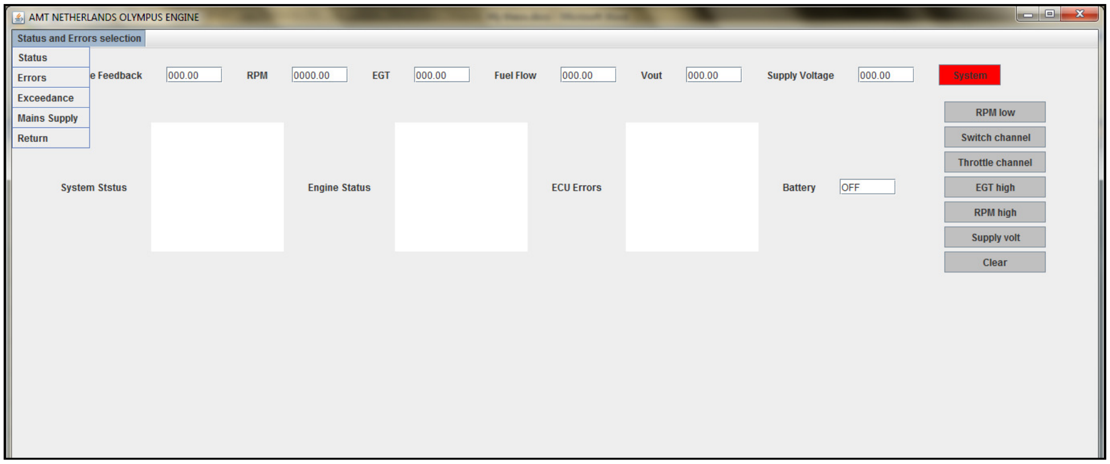


Figure 60 – General test harness GUI

6.4 Validation

Unit testing was applied wherever possible. Methods with explicit return values were tested with JUnit generated test harnesses. Methods which required TCP connection were unit tested with the use of self-designed test harness, specifically one acting as a TCP client and another as a TCP server. There were also several methods which were only possible to be tested during integration of the methods and the engagement of the General Test Harness, due to dependencies on other elements. Such methods were the main functions, methods attaching shutdown hooks or void methods writing to text fields and areas. The constructors were tested by observation of the instantiated objects. The unit test coverage of the involved modules is shown in Table 7.

Table 7 – Unit testing of EOCS modules

Module	Total methods	JUnit	Self-designed test harnesses	Integration tested
EISI	18	8	2	8
MEOCS	15	2	2	11
Gen. Test Harness	10	4	3	3

After integration of the functions and the completion of the General Test Harness, the EISI and the MEOCS underwent control flow testing individually, with the test cases defined with the estimation of the cyclomatic complexity, based on the derived control flow diagrams (Appendix C, Sections C.2.1- C.2.2). Consequently, the integration testing followed, where the EISI and MEOCS were integrated and operated against the General Test Harness, using all the previously derived test cases. Finally, the same test cases were also applied with the EIS and the actual gas turbine engaged.

However, in order to obtain 100% decision coverage, the reception and presentation of all the possible engine and system outputs should be verified as well, although the individual methods for these functions were thoroughly tested during the preceding unit testing. For this reason, all tests applied during the EIS validation (Appendix C, Section C.1) were reapplied. Tests not possible to be simulated with the real engine (ECU warnings, system messages and exceedance values) were recreated with the help of the General Test Harness.

Additionally, as throttle settings ranged from 0 to 100, the possible engine output would vary enormously. Hence during the validation runs of the gas turbine the operation was divided into groups of throttle setting values: throttle 0%, throttle 20%, throttle 40%, throttle 60%, throttle 80%, throttle 100% (partition testing). Boundary testing was also conducted for the following indications of the engines, in order to verify that the value depicted on the MEOCS GUI was expected and equal to those on the EIS panel and the engine EDT:

- Throttle feedback
- EGT readings
- RPM readings
- Fuel Pump Voltage
- Supply Voltage

6.5 Results – Discussion

The objectives of this round were accomplished and successful interaction between Java and LabVIEW programs was established. The initial approach considered building the EIS in a LabVIEW DLL that would be loaded in a dedicated thread from the Java EISI, with the application of JNA. During the software process, the operating system of the connection PC was upgraded to Windows 7 64 bit and Java 64 bit was installed. As a result, the DLL could not be loaded, as it was built in 32 bit representation. It was then considered to upgrade LabVIEW from 2011 32 bit to 2012 64 bit, but research in the website and forums of National Instruments revealed that even with a 64 bit version of LabVIEW, the included VISA modules could only be built in a 32 bit DLL, which could not be loaded by Java 64 bit runtime. Although a 32 bit version of Java could be installed, it was decided to make an approach which would be more compatible for 64 bit versions of Java and LabVIEW, with the least effects to the already designed modules. Thus, the EIS was built in an exe file and was executed as a Process object by Java. Since no data was required to be passed to the DLL during the invocation, this concept performed sufficiently. The only drawback was the fact that the child process needed to be destroyed explicitly when the EISI was closed. This was obtained with the attachment of a shutdown hook that destroys the EIS child application upon closure of the parent program.

The inadequate recognition of the dependencies between the software components as an identified risk was avoided and appropriate communication was established between them. There were no implications that would give rise to the cost of the project, as also indicated by the risk assessment of the phase.

It was attempted to achieve the highest possible coverage of designed methods with JUnit, although this was not totally accomplished, as shown in Table 7. It was inevitable due to the inclusion of sockets exchanging information constantly, or other specificities, such as shutdown hooks. However, self-designed test harness contributed to overcome the problem. Moreover, after completion of the General Test Harness, it was possible for all methods to be validated, even if unit testing could not be applied without dependent programs engaged.

The validation of each module was thorough and 100 % statement and decision coverage was obtained, guided by the control flow diagrams of the developed software modules. The cyclomatic complexity of each module - derived from the control flow diagrams after integration of the comprising methods - had a value of 10 or less (Table 8), which is conventionally considered as the maximum desired value, in order to retain complexity of designed code under control.

Table 8 – Cyclomatic complexity of designed software modules

Module	Cyclomatic complexity
EISI	7
MEOCS	10
General Test Harness	6

Although several individual methods in the EISI and the GTH had high cyclomatic complexity values (due to switch conditions with many potential values, as a result of the engine and system output messages), these were treated as black boxes in the final control flow diagrams. Their potential outcomes had all been exhaustively tested during the unit testing phase.

The analysis for possible memory leaks for EISI and MEOCS has shown that after initial high objects creation, the surviving generations stabilised, as shown by the profiling tool of Netbeans, which was applied. The initial escalation was caused after the RMI invocation of the relevant Threads. The programs were executed for a period of time near an hour in order to observe memory consumption. The EISI object creation escalated upon start but later on stabilised (Figure 61). The heap size allocated was observed to be around 150 MB, also stable during runtime (Figure 62). The MEOCS also showed an escalation of surviving generations, as the GUI and continuous updating of text fields and areas causes object generation. However, it was stabilised after a while and the garbage collection functioned satisfactory at default settings (Figure 63). The heap size and usage also appeared stable (Figure 64). Initially observed high numbers and increasing rate of surviving generations were reduced by setting String and other objects to null values, after their use in While loops. Null objects are more likely to be collected during regular garbage collection, rather than remaining as surviving generations. It can also be observed by the profiling graphs that the time spent in garbage collection – purple line - (Figure 61, Figure 63) was very low, preventing performance degradation (the JVM ceases other functions while performing garbage collection).

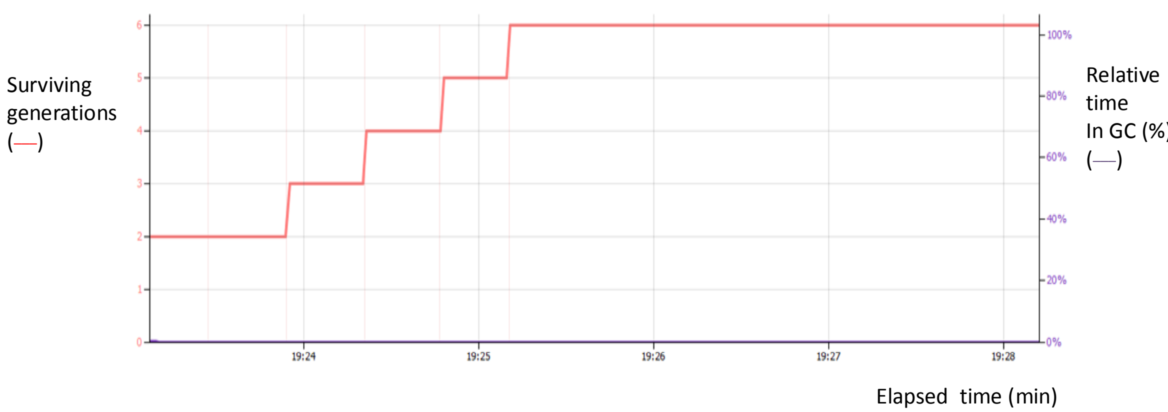


Figure 61 – EISI surviving generations

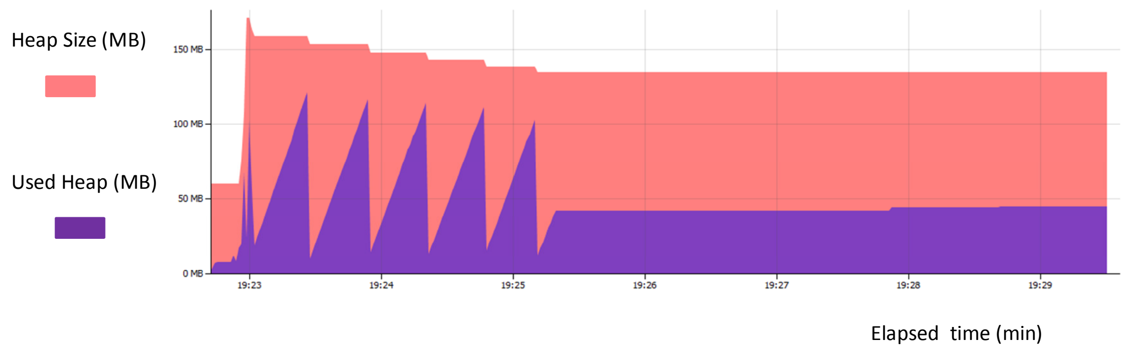


Figure 62 – EISI heap size and usage

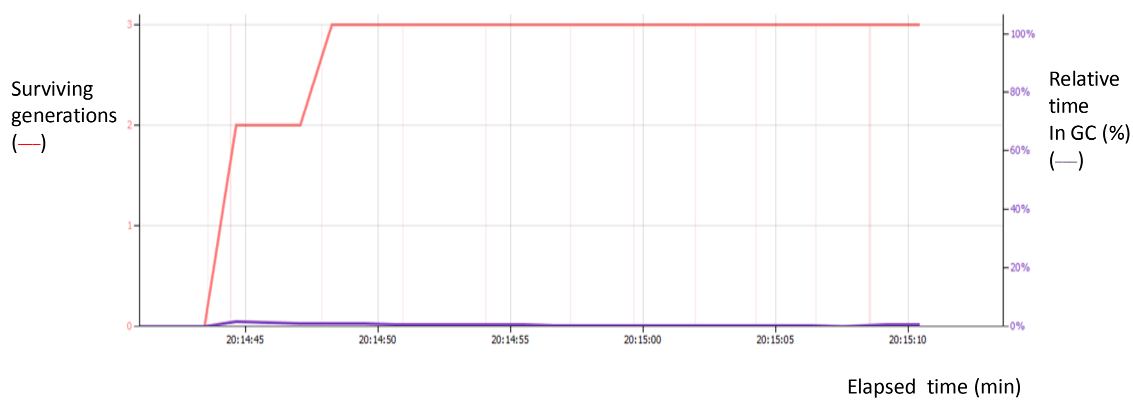


Figure 63 – MEOCS surviving generations

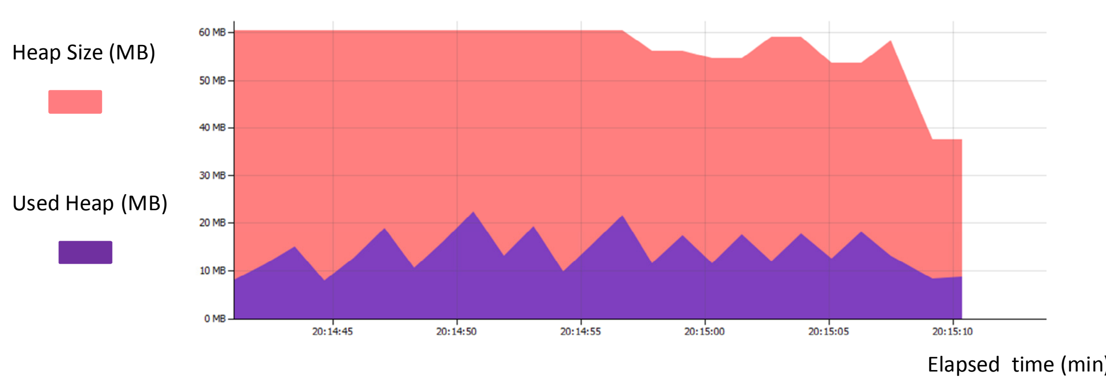


Figure 64 – MEOCS heap size and usage

The estimation of the command latency experiment was conducted from 3 different computers: the local PC connected to the engine and two other buildings of the University campus, located approximately 1 and 1.5 kilometres away from the test house where the gas turbine was installed, within the same LAN. The latency from the local PC was negligible (less than 0.5 ms). For the other two locations, the data showed dispersion around 0.2 sec. There was no noticeable difference between the two locations, or even during the different time slots that the measurements were taken (Figure 65, Figure 66, Figure 67).

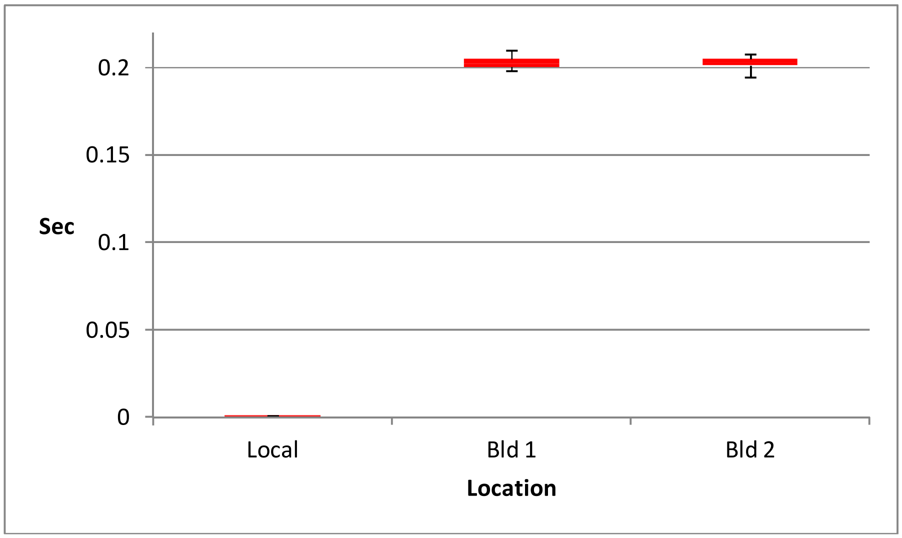


Figure 65 – Box plot of command latency for time slot 09:00 – 10:00

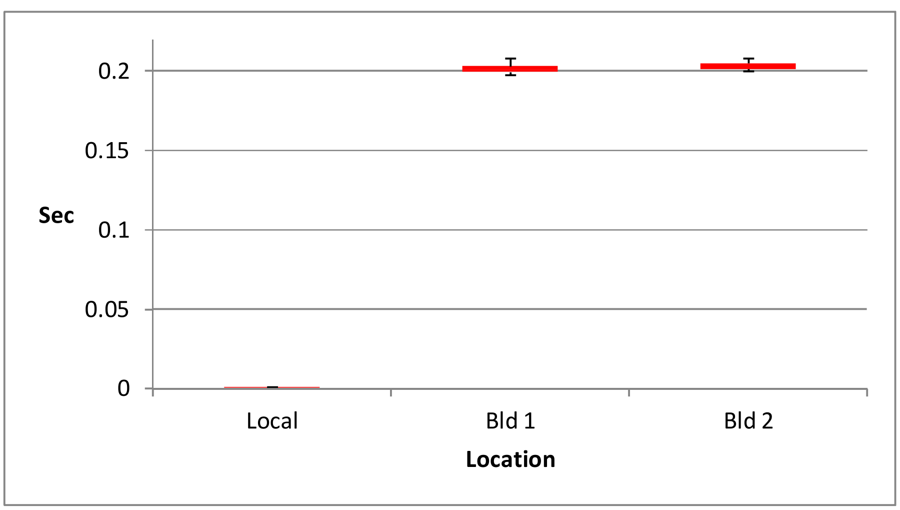


Figure 66 - Box plot of command latency for time slot 13:00 – 14:00

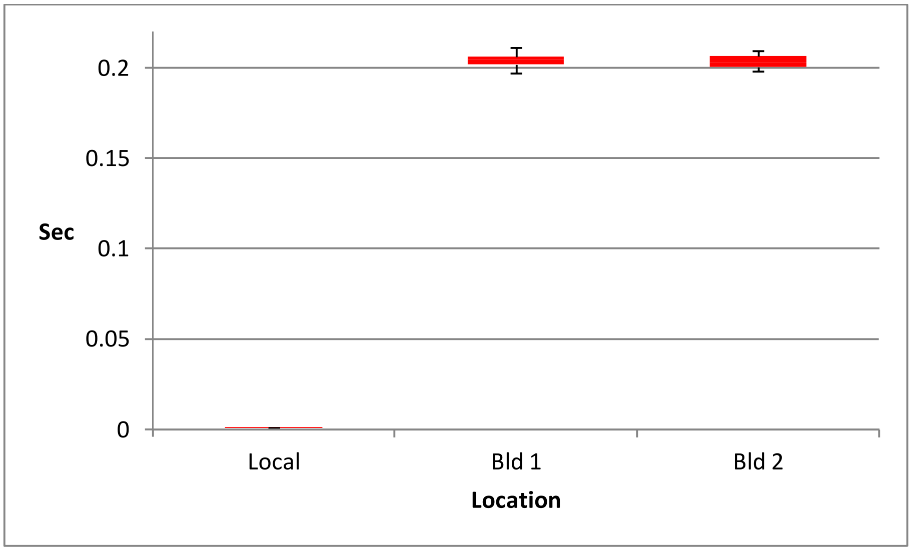


Figure 67 - Box plot of command latency for time slot 15:00 – 16:00

The latency was inevitable, due to physical impedance of the network. However, it was noticeable and action for prevention of potential undesirable conditions was regarded as necessary. This issue was addressed in chapter 8.

The final element to be discussed for this round was the configuration of the engine output. This was obtained by the implementation of several methods, each one of which receives the EIS output and identifies the appropriate values in the expected order. The implementation of these methods must not be concrete if other engines were also installed in the system. That is why they were defined in an interface as abstract methods. The method definitions will remain generic but their concrete implementation will be unique for each type of gas turbine included. The generic capabilities and prospective of the project will be discussed in chapter 10.

7 ADDITIONAL FEATURES

This chapter describes the approach and implementation of round 3 of the Spiral process. It presents additional functionality to the system, in order to render it safer and friendlier to the user. By the end of this phase, a completed LAN prototype was available.

7.1 Methodology

The methodology applied in chapter 7 was guided from the objectives, shown in Table 9.

Table 9 – Objectives of round 3

Addition of live camera image – examination of intercom capability
Design of a reliable Trouble Shooting System on gas turbine side computer
Acquisition of fuel flow, mains supply voltage, ambient temperature and ambient pressure
Installation of additional warning and safety features on the engine side
Completion of a fully functional LAN prototype

7.1.1 Ambient parameters and mains power supply – Live image

The live camera image was captured by a simple web camera with the use of JavaCV [56], an open source API which allows the implementation of openCV, in Java environment. Opencv is written in optimized C/C++ and it was designed for computational efficiency, with a strong focus on real-time applications [76]. UDP protocol was used for the transmission of the image through the network, in order to obtain fast transmission. UDP is often used for video, as it preferred to have a few packets missing or out-of-order rather than to have the full stream with lag and delay. Accuracy of the transmission was not crucial for the live image. The loss of a few frames would not have any impact as if a command was corrupted hence no additional work for the transmission quality was required. The differences between UDP and TCP were discussed thoroughly in section 3.7.5.

Ambient temperature acquisition was obtained with a K type thermocouple connected to a NI 9265 C series module. Ambient pressure was acquired with the use of a digital barometer, transmitting the information through a serial bus to the connected PC. To monitor the mains power supply, an AC to AC transformer was used in order to reduce the alternating voltage from ±240 Volts to ±9 Volts, rendering it safe to be captured by the NI 9215 C series module for analogue input. The processing of the ambient and power supply data takes place in the ACAS software module, which was implemented with LabVIEW and VISA (Virtual Instrument Software Architecture) – these tools were also used for the EIS module. For the ambient pressure reading, a reusable LabVIEW VI previously designed by the Department for an older project was used. The value acquired from this VI is then provided to the ACAS.

Fuel flow acquisition was obtained with the installation of a Floscan 201-A6 fuel flow transducer. It requires a 12 Volts DC power source, provided by an AC to DC converter and it is suitable for small flows (0.3 – 3 GPH). The signal produced by the flow transducer is an open collector transistor output. The sensor will pull down to 1.0 volt with a 10-15k ohm pull up resistor installed [43]. According to the recommended by the manufacturer connection (Figure 68), the resistor was installed between the flow signal and the additional 5 Volts DC input, provided by an available wire from the ECU of the engine. The K factor was used as provided by the manufacturer without further calibration. Fuel flow was not considered as one of the highest priority readings of the gas turbine for safety purposes, thus accuracy was not very important at this stage. The processing of the fuel flow transducer output was obtained with a NI 9401 C series module for digital input/output.

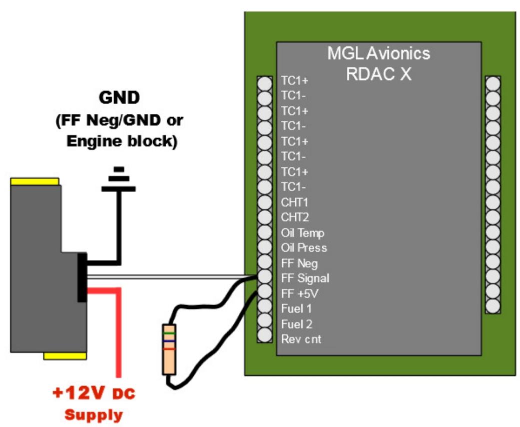


Figure 68 – Suggested connection of Floscan 201-A6 Fuel Flow Transducer [43]

7.1.2 Peripheral features

An addition to the existing EDT (Engine Data Terminal) was designed to accommodate a panel with warning and indication lights. Appropriate LED bulbs must illuminate to indicate to the observer of the EDT that the system is ready to function remotely. In case of an error signal produced from the connected computer, warning lights are illuminated, notifying the on-site observer to take manual control. Manual control can easily be achieved by changing the position of the signal switches and the notification switch on the junction box. The latter informs the remote user that manual control has been established. In the extreme situation of a malfunction of the junction box, the wiring of the signal channels of the ECU can easily be detached from the junction box and re-connected with the original plugs of the manual control box wiring. Extra safety is provided with the installation of a master switch at the position of the adjacent computer observer station. The switch should receive a signal from the EIS and if closed, it returns an indication of the signal, allowing for remote commands to be accepted. Thus, if the gas turbine is under service or not in a situation to run, the master switch set to off position will prevent any remote operation, even if the application is running.

All the above signals were analogue DC voltage signals generated by the NI 9263 C series module already installed and the circuitry was configured with gauge 22 wiring. The LED bulbs were installed in series, in accordance with LED installation instructions [86].

7.1.3 The Trouble Shooting System module (TSS)

One of the most vital components of the whole application is the Trouble Shooting System (TSS). This system monitors the gas turbine and transmits messages to the user about relative or absolute limits exceedance. In the case of the Olympus engine, the critical parameters monitored are the RPM, the EGT and the mains power supply to the connected computer and hardware. Relative limits were derived from the gas turbine control systems application of alarm rules, in order to detect values above certain limits during the engine operation. The rules applied in this case were the Western Electric rules, which are the most common in aviation control systems [78]. According to these, there are four unusual patterns of data points (Figure 69):

- The Instability pattern
- The mixture pattern
- The stratification pattern
- The trend pattern

More details of the above patterns can be found in section 3.2.4. The most noticeable for the present study was the instability pattern, which means the existence of data points outside the control limits. They divide the control band into three zones where zone A is the area enclosed within 2 and 3 standard deviations (σ) of the parameter, zone B, which stays between 1σ and 2σ and C, between ±1σ. A guideline for the alarm generation is any point beyond zone A, 2 out of three consecutive points in zone A, 3 out of five consecutive points in zone B and 8 consecutive points on the same side of the centreline [78].

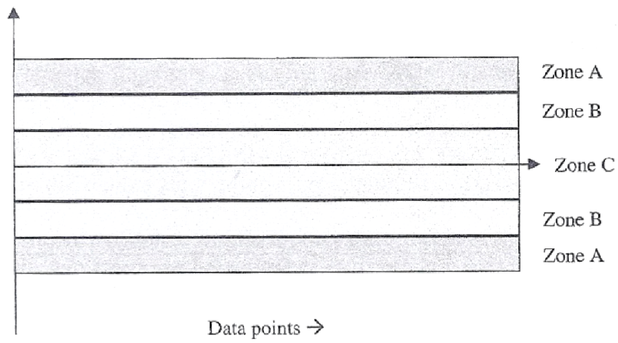


Figure 69 – The Western Rules [78]

The nominal values of RPM and EGT were established after experimental sampling with the gas turbine running throughout the whole throttle range. The standard variation of EGT and RPM were determined by variation of the throttle setting from 0% to 100%, in increments of 1%. Corrections for ISASL conditions were applied to the readings prior to processing [145]. Selected sample sets from the obtained data for each throttle setting were examined for distribution fitness. The fitness test was accomplished with the application of ExpertFit, a distribution fitting software tool developed by Averill M. Law and Associates. ExpertFit can determine automatically and accurately which of 40 included probability distributions best describe the given data [85].

As observed from the general risk analysis, the loss of power supply to the computer at the gas turbine side, poses one of the main hazard threats to the system. This situation was mitigated with 2 actions: installation of a power stabiliser and by monitoring the mains power output. The stabiliser ensures the prevention of abrupt shut down in the occasion of power loss and allows for stable operation for a few minutes. In addition, by monitoring the power supply, the end user receives an indication on his caution panel and if he does not quit the application within 3 minutes, the engine will be shut down by the TSS and the program closed automatically.

The TSS was written in Java SE, enabling easy data exchange with the other Java modules. Java Exchanger objects were used to synchronise the TSS and the EISI, so that they can exchange the necessary data: ambient and peripheral information towards EISI, and RPM with EGT towards TSS. TSS was also investigated for memory leaks and memory consumption with the use of the Netbeans IDE profiler. After integration with the EISI and MEOCS, the latter two were re-examined for potential memory leaks.

Moreover, all Java modules were inspected for several quality metrics, as described by Oliveira et. al. (2008) [109]. The parameters considered included:

- Number of Lines of Code (LOC)
- Average instability (I), which implies encapsulation, reusability and maintainability
- Lack of Cohesion of Methods (LOCM) that indicates cohesion in classes
- Average Cyclomatic complexity (VG), Nested Block Depth (NBD) and Weighted Methods per Class (WMC), which indicate complexity of the designed modules

7.1.4 Validation Methodology

Validation included unit testing of individual functions (JUnit) [52], inspection of the code, and control flow testing of complete functions. LabVIEW test Unit Framework was applied for unit testing of the LabVIEW functions and the physical signals were also measured with an oscilloscope. Integration testing was accomplished after integration of the individual modules, with the use of the General Test Harness. Control flow testing was applied during the integration testing, based on the derived control flow diagrams of the designed software components and the cyclomatic complexity suggested by Tom McCabe [147].

7.2 Risk Assessment

The process has been examined for underlying procedural and physical risks. The physical risks were analysed by FMEA and FMECA. The NI hardware components had a given MTBF. For the electric and electronic components that did not have published MTBF, once they had been purchased from reliable manufacturers and complied with the Mil– HDBK - 217F standard, it has been assumed that they had similar MTBF standards. Any part that had been modified in the lab to match specific needs of the project has been assumed to have failure frequency of one level higher (Appendix B, Section B.5).

The highest scoring physical risks reached the value of 9 (undesirable) according to FMECA and they involved looseness of cable connections and power supply to the individual components of the system. For these reasons, special care has been taken to secure the added wiring and devices. The issue of power supply instability or loss has been addressed in chapter 5. However, after completion of the TSS, the related risks will be alleviated.

The cost was also addressed in the procedural risk assessment, as this round involved the procurement of new devices and components which could easily introduce cost that would deviate from the initial budget. In order to prevent unnecessary cost increase, thorough market research took place along with very careful study of available specifications and manuals (Appendix B, Section B.5.1). Other elements considered were validation, software dependencies and hardware selection. The necessity for detailed physical risk assessment was also identified in the procedural risk assessment, leading to the FMECA analysis described previously.

7.3 Analysis, Design and Implementation

This section describes the analysis, design and implementation of all the hardware and software modules addressed by the objectives, the System Requirements and risk analysis for round 3 of the Spiral process. Sub – section 7.3.1 presents the Ambient Conditions Acquisition Software (ACAS) design and all the additional functions in the Engine Interface Software (EIS), sub - section 7.3.2 describes the peripheral hardware installation and sub - section 7.3.3 outlines the Troubleshooting System (TSS) design and implementation.

7.3.1 The ACAS – Addition of other modules and adaptations

Six new functions were implemented in the EIS.vi, 2 from which perform independently. One of them (UtilitiesSignalGeneration) generates the 10 Volts DC that is supplied to the two newly introduced switches: the PC observer's position switch and the manual acknowledgement switch at the junction box. When these two switches are closed, signal feedback is provided to the newly introduced function (UtilitiesAcquisitionSignal). The first switch returns a 10 Volts DC signal to indicate that the observer is in position hence the system is ready. The latter switch returns 10 Volts DC signal to indicate manual control reception from the engine EDT observer. The UtilitiesAcquisitionSignal function is associated with the TCPClient function, which receives the signal acquired from the PC observer's On/Off switch. The Manual Acknowledgment switch signal is supplied to another newly introduced function (ControlStatusConfiguration), which configures the final message that contains the status of the operation control. This can either be NORMAL, LOCAL OVERRIDE, MANUAL or MANUAL & LOCAL OVERRIDE. Additionally, the signal provided to the Manual Acknowledgement switch is branched to provide a permanent indication of signal existence to the EDT panel.

A function for the generation of 10 Volts DC signal to the new indication LED's on the EDT was also implemented (EDTWarning function). It generates a signal when an error of the command signals or the engine output reading function occurs. Finally, the FuelFlowRead function was implemented to read the pulse signal generated by the fuel flow transducer. After the signal is interpreted it is provided to the TCPClient to be transmitted through the network. Variables between the functions are passed via local variables and are assumed public in the UML approach of the class diagrams design (Figure 71).

The other group of LabVIEW components that was implemented included 3 VI's (Figure 72). The Barometer.vi and GeneralRead/Write.vi were reusable modules previously designed by the Department. AmbientSensing.vi was newly implemented (Figure 72). The latter includes the MainsSupplyInterfaceSystem (MSIS) function, which reads the AC voltage signal from the mains supply, after conversion to ±9 Volts DC. In order to disregard any temporary instabilities of the system, this function generates an alert only if the absolute value of the input signal has been under 1 Volt for more than 3 seconds. The alert is then provided to the TCPClient function to be transmitted.

A technical difficulty was discovered at this point, as the DAQ unit could not create analogue input virtual channels in different VI's. Hence, in order to overcome the problem without additional equipment, the mains supply signal was acquired from the UtilitiesAcquisitionSignal function of the EIS.vi and provided to the MSIS via a LabVIEW global variable, in order to adhere to the concept that ambient and peripheral data should be acquired by the same LabVIEW VI and transmitted within the same package of data.

The AmbientTemperatureSignal function acquires the signal of the K – Type thermocouple through a dedicated hardware module (NI 9265 module) and virtual channel. The ambient pressure is acquired by the reusable Barometer.VI and GeneralRead/Write.VI and provided as serial input directly to the PC via a USB interface. Consequently, temperature and pressure are passed via local variables to the TCPClient for transmission (Figure 72). The complete Hierarchy of the LabVIEW components included in the application is shown in Figure 70.

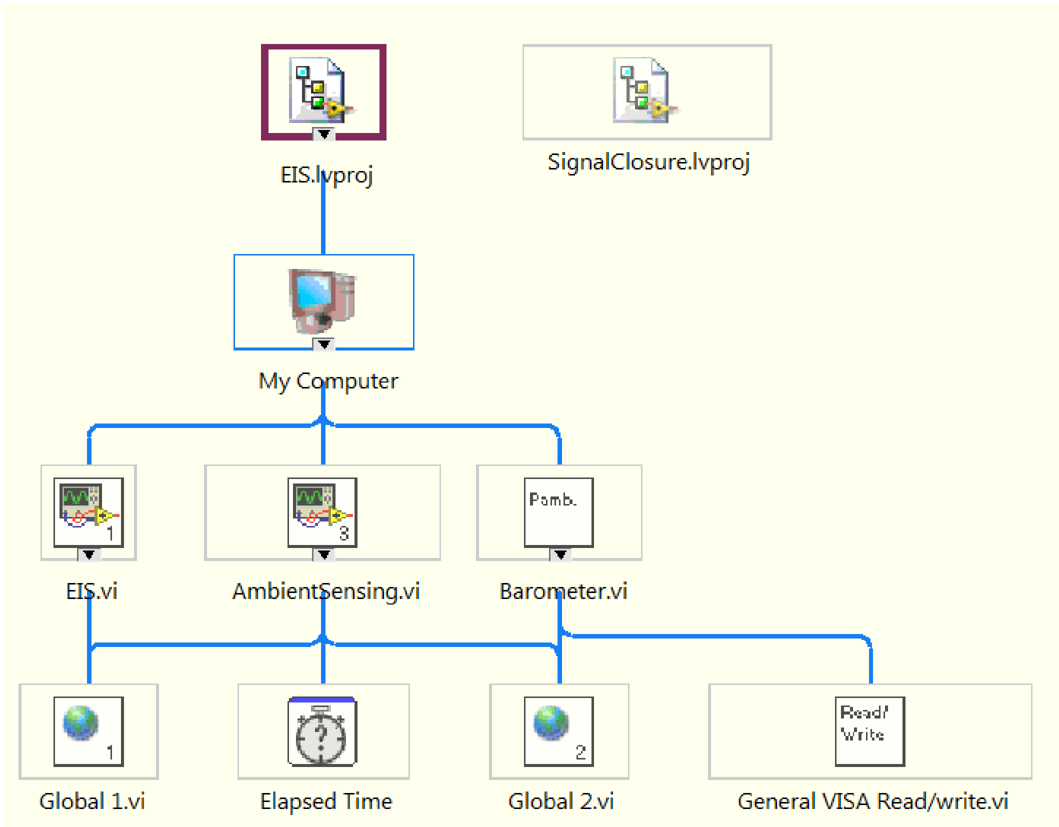


Figure 70 – LabVIEW projects and VI hierarchy

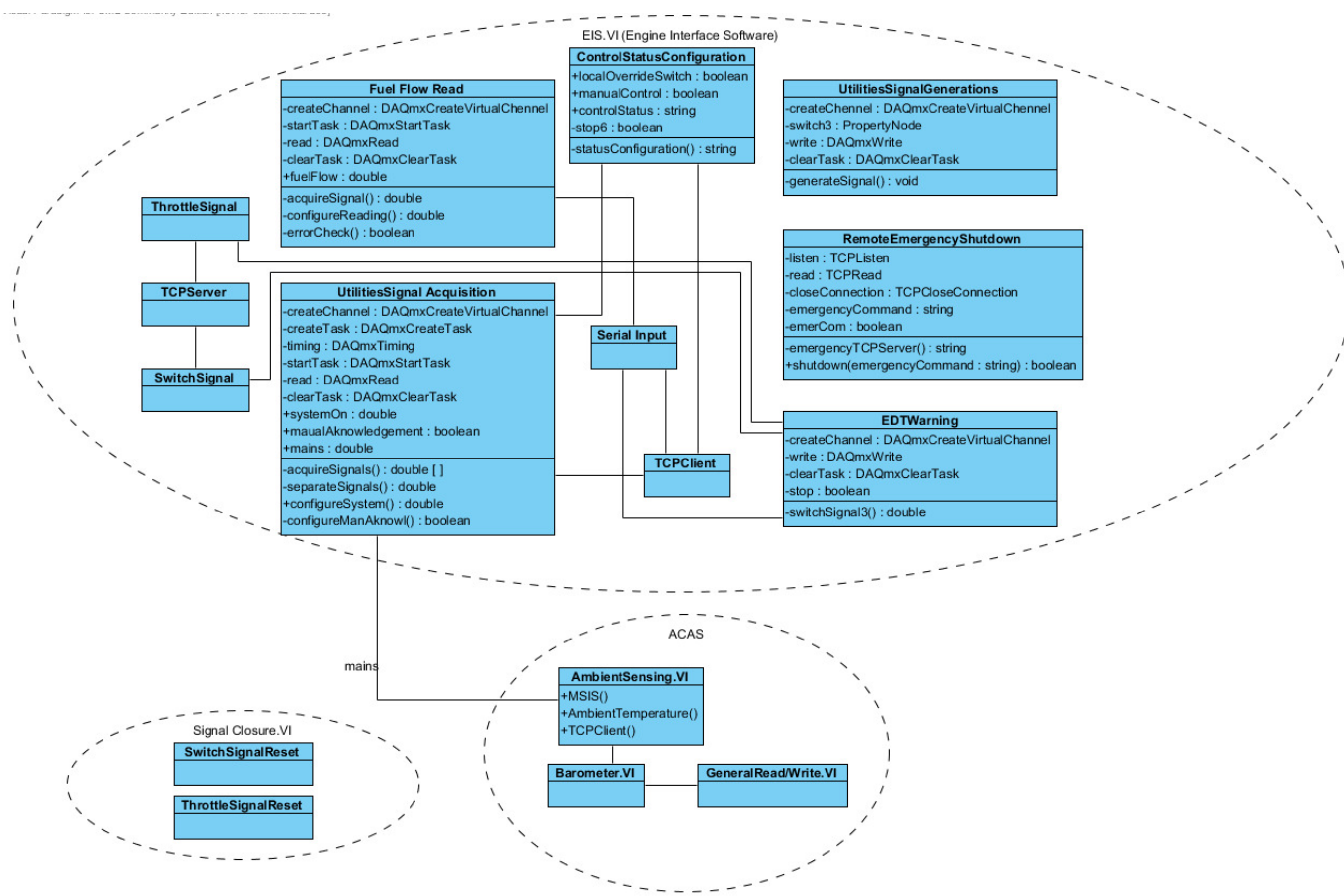


Figure 71 – The EIS and ACAS association UML class diagram

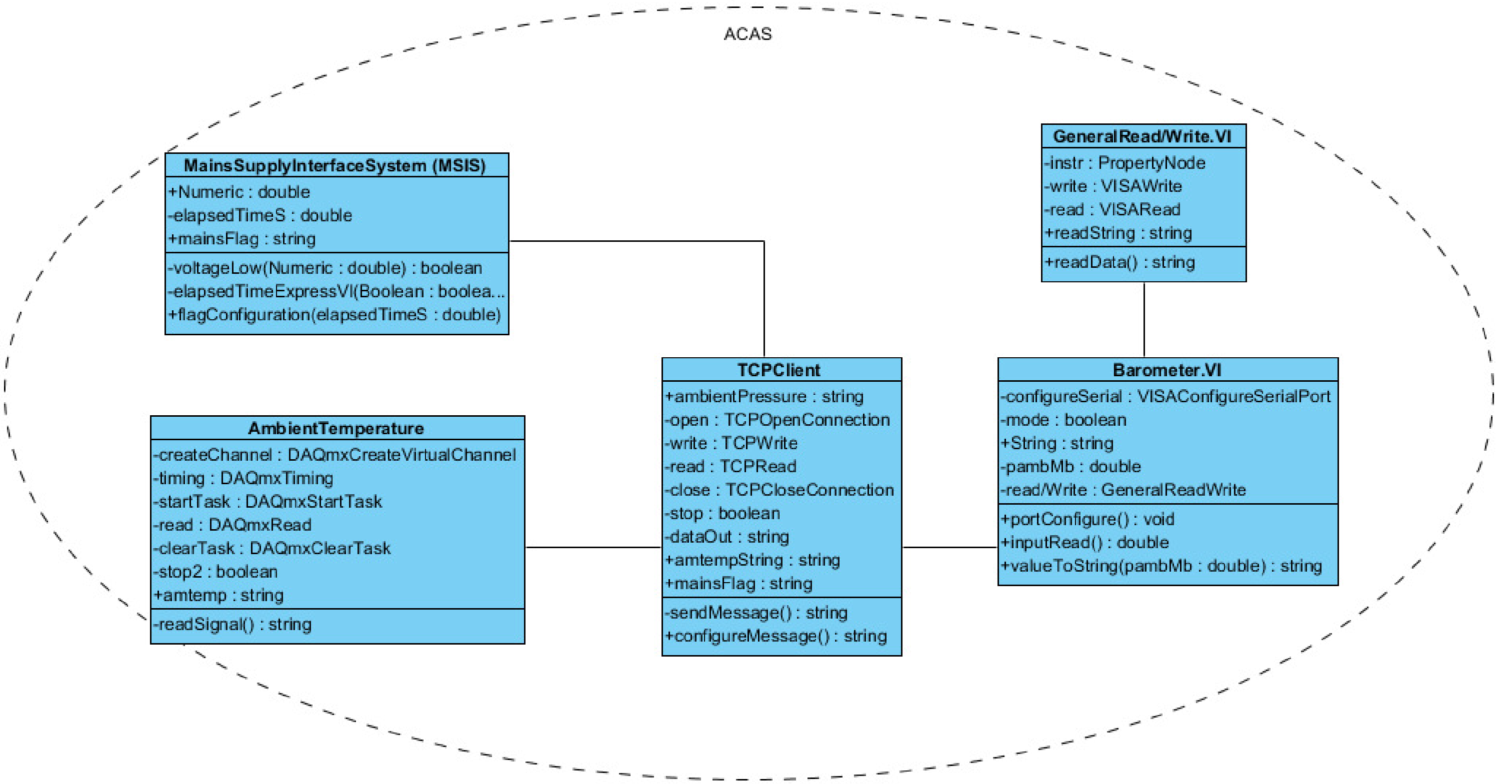


Figure 72 – The ACAS UML class diagram

A new thread was added to the EIS Interface (EISI), dedicated to the camera invocation and capturing of live image. The thread is instantiated upon the call to class CamCapture(), which has imported several JavaCV libraries. The web cam is started with a call to the the start() method of the grabber attribute, which essentially is an openCV FrameGrabber object. The grab() method of the grabber is then called to save the image as an openCV IplImage object. Consequently, the image is converted to an array of bytes and transferred on top of UDP to the Main Engine Operation Control Software (MEOCS).

Accordingly, another thread was added to the MEOCS side for the reception of the live video stream. It is instantiated upon the call to the class CamReception(), which has imported the JavaCV library as well. The image is continuously received within a while loop as an array of bytes and consequently converted to a BufferedImage object before being displayed on a JavaCV Canvas frame. The Canvas frame has been configured to always remain on top of other open windows in the user's screen, ensuring that the live video stream from the engine is always visible. The exit upon closure feature of the frame has been disabled so that it may only close when the main window of the user interface is closed.

Two classes were added in the main package of the EISI to accommodate the data reception from the TSS. The TSSCall class loads the static central method of the TSS and runs on a dedicated thread. The other class introduced - also in a dedicated thread - was the TSSInfo(), which receives the data from the TSS on top of TCP and transmits them to the MEOCS. Moreover, an additional TCP socket was introduced in the existing ServerEIS() class to transmit the critical parameters to the TSS (Figure 73). The MEOCS was accordingly configured with a dedicated thread instantiated by one added class, to receive the TSS info and depict the information on the user interface. The user interface is only instantiated once and then passed as an argument to the two threads which receive data from the EISI, in order for the appropriate fields to be updated constantly (Figure 74).

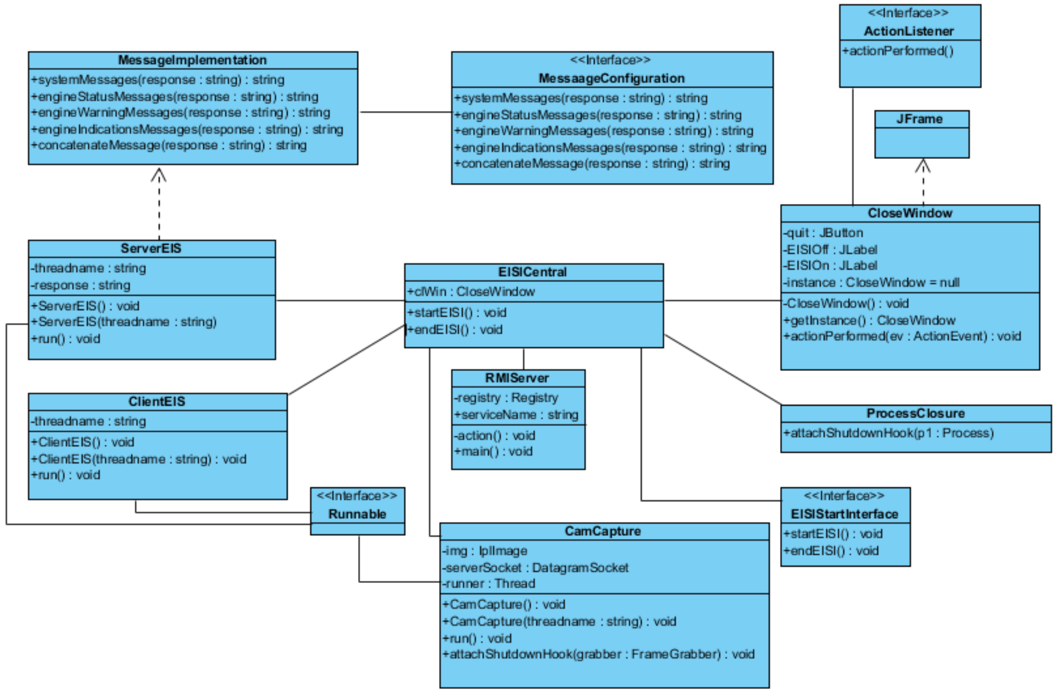


Figure 73 – EISI configured for the TSS and camera reception

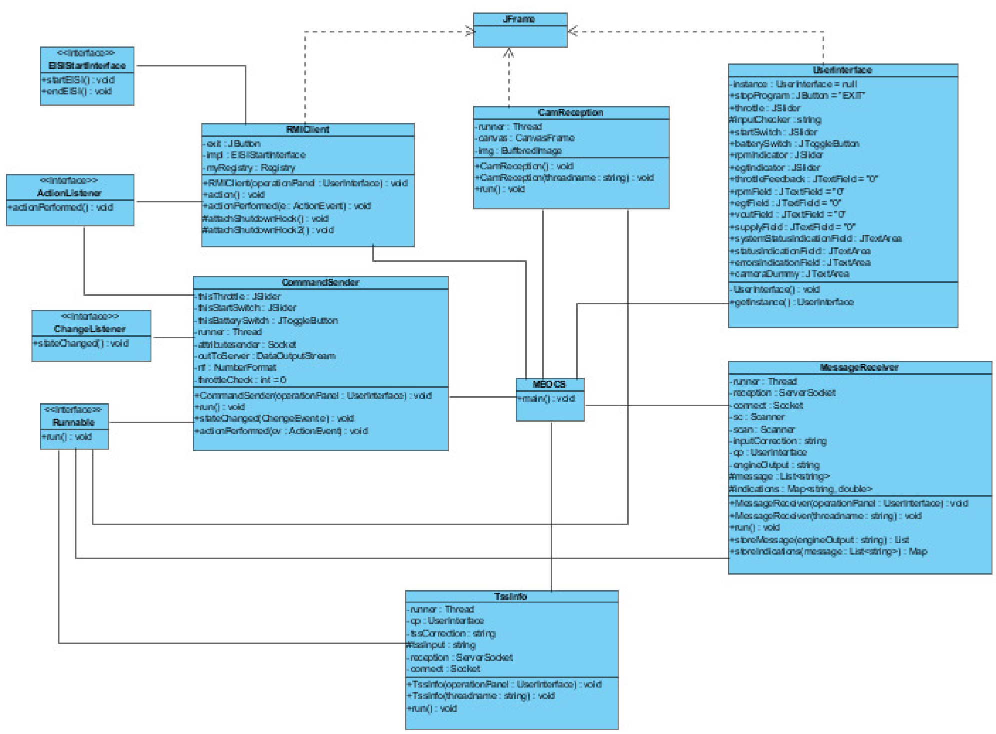


Figure 74 - MEOCS configured for the TSS and camera reception

7.3.2 Peripheral Hardware

The peripheral hardware installed consisted of the following:

- An AC to AC converter connected directly to the NI 9215 module, providing ±9 Volts AC voltage.
- A 12 Volts AC to DC converter, connected directly to the Fuel flow transducer
- Two on/off switches: one for the PC observer's position to enable or disable command signals generation from the system and the other for the engine EDT observer to indicate that he has recovered manual control of the engine operation.
- Four 3mm LED indicators. Two of them must illuminate when the system is running. Upon signal loss, either due to power supply loss, cDAQ failure or computer failure, they will cease to illuminate. The other two are illuminated upon signal error, which implies that the engine observer must switch to manual control. The LED pairs were installed in series with a resistor of 270 Ohm connected after each one of them, as indicated by the LED wizard (Figure 75). The Wiring diagram of the LED's, the switches and the NI modules is shown in Figure 76.

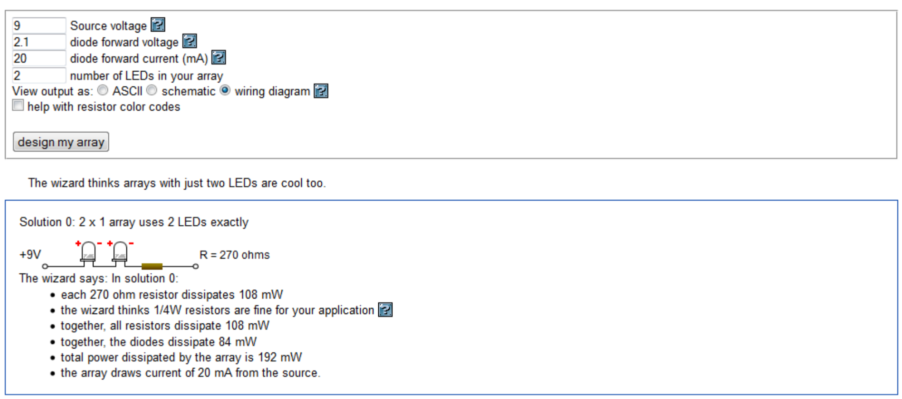


Figure 75 – LED configuration

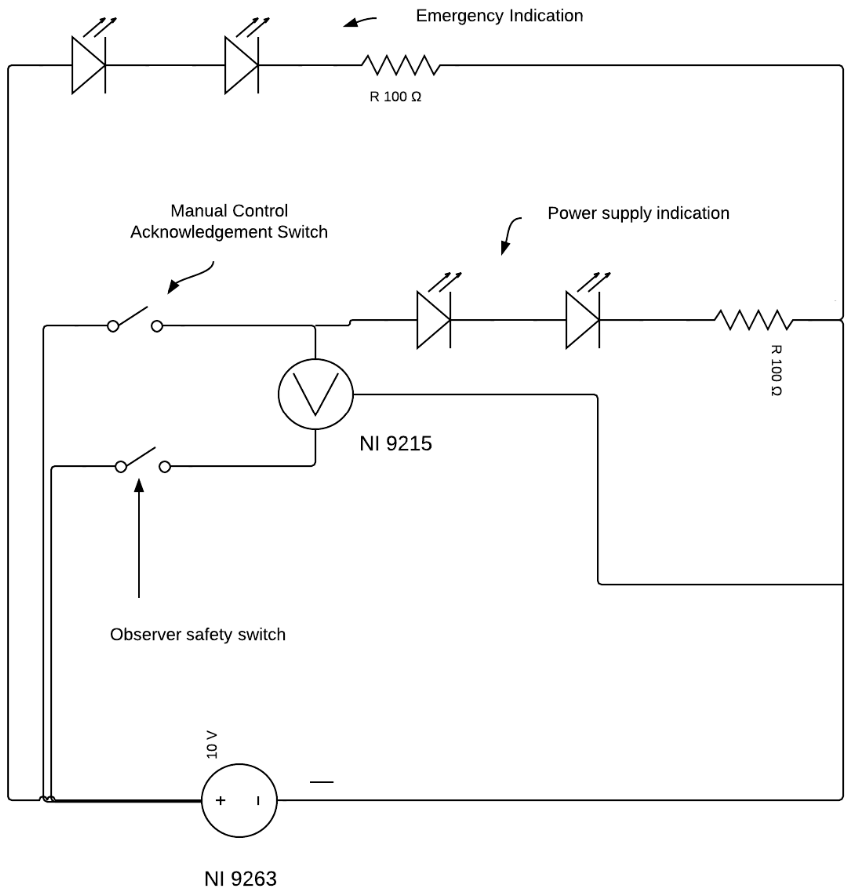


Figure 76 – Peripherals wiring diagram

7.3.3 The Troubleshooting System (TSS)

The TSS architecture was derived from a simple use case analysis with the interacting modules as actors. The analysis was based on the required data exchange between the already existing software components and the TSS (Figure 78). A prerequisite for the successful interaction was the existence of a text file that includes the nominal values of the critical parameters for each throttle setting, ranging from 0% to 100%. The data was collected and then assessed with ExpertFit for distribution fitness (Figure 77). Six data sets were selected for each group: EGT and RPM. The groups were defined by the throttle setting value from 0% to 100% in increments of 20%: 0%, 20%, 40%, 60%, 80% and 100%. The distributions with the best fitness results were Normal and Weibull. However, Normal was by far mostly present (Table 10 – ExpertFit distribution fitness evaluation) hence the average values and standard deviations can safely be estimated by the following formulae:

$$\mu = \frac{\sum_{i=1}^{N} x_i}{N}$$

Equation 4 – Average value of statistical sample

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x-\bar{x})^2}{n-1}}$$

Equation 5 – Standard deviation of statistical sample

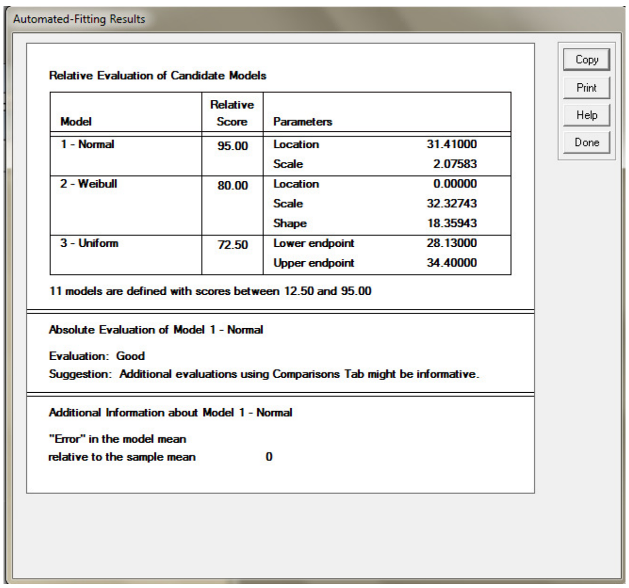


Figure 77 – Example of ExpertFit distribution fitting evaluation

Table 10 – ExpertFit distribution fitness evaluation

Distributions	Best Fitting Results	Evaluation Result
Normal	9	Good
Weibull	2	Good
Weibull & Normal (equal score)	1	Good

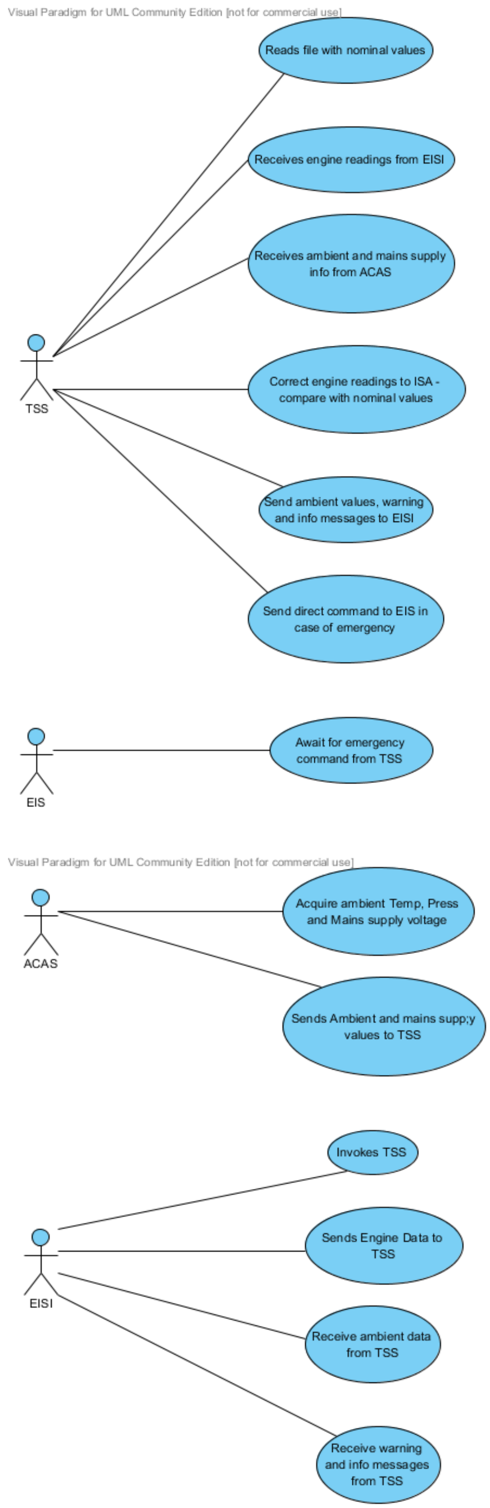


Figure 78 – Use case analysis for the design of the TSS

The basic concept of the TSS was to receive ambient and peripheral data directly from the ACAS, correct the critical parameters (RPM and EGT) to ISASL conditions (Equation 6, Equation 7, Equation 8 , Equation 9), compare them to the stored nominal values and then send the information to the EISI (Figure 79).

$$\delta = \frac{P_0}{101325Pa}$$

Equation 6 – Referred temperature

$$\theta = \frac{T_0}{288.15K}$$

Equation 7 – Referred pressure

$$EGT_{cor} = EGT/\theta$$

Equation 8 – EGT corrected (referred) to ISASL

$$RPM_{cor} = RPM/\sqrt{\theta}$$

Equation 9 – RPM corrected (referred) to ISASL

The necessary data from the engine output is provided to the TSS from the EISI, which is connected with the EIS. To obtain the data exchange, the EISI needed to include a new TCP server to receive data from the TSS and a TCP client to transmit information to the TSS. Respectively, the TSS required a TCP client and server to connect with the EISI, along with an appropriate TCP server to interact with the ACAS (Figure 80).

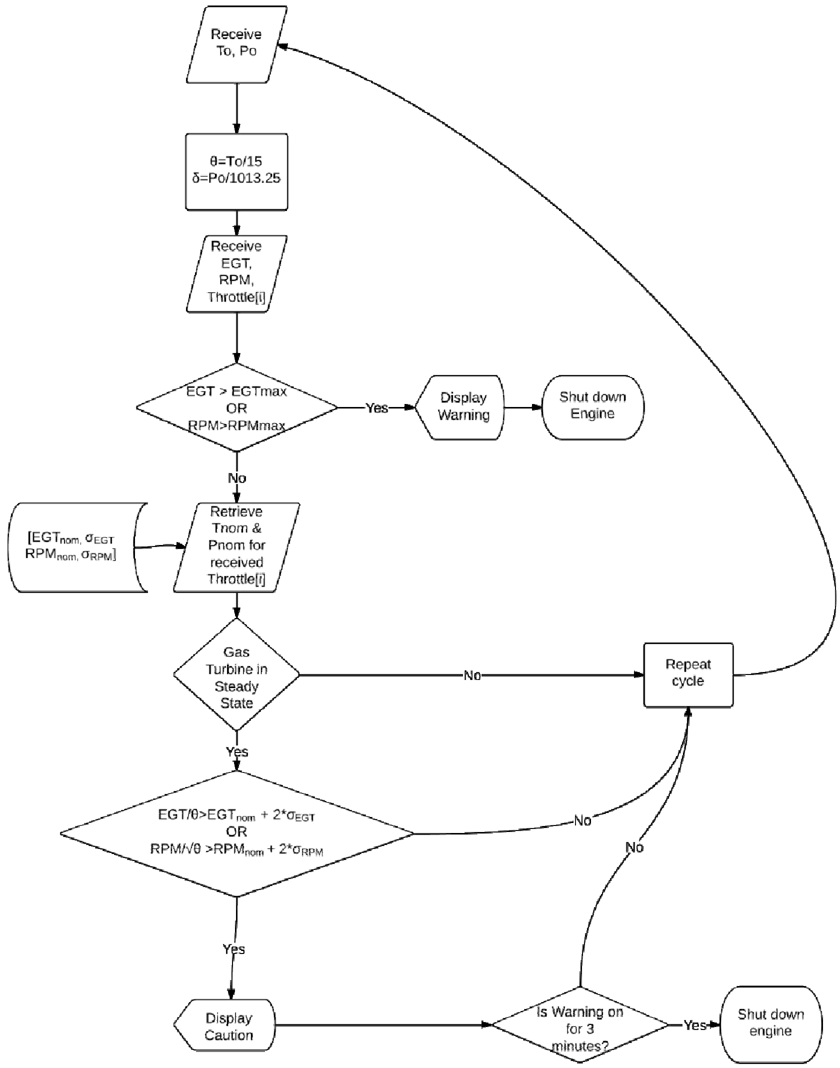


Figure 79 – Logic of the TSS

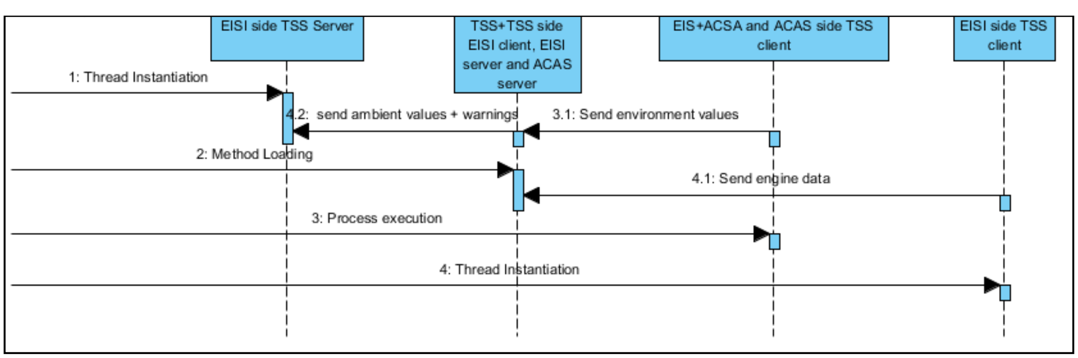


Figure 80 – Sequence diagram of the TSS and the surrounding software modules

The proposed design for the TSS included a central class with a static function, which is invoked by the EISI and consequently initiates the sequence of the TSS, a class to read in the nominal data file, a thread to receive the ambient and peripherals data, a thread to receive engine data, a thread to connect directly to the EIS and shutdown the engine upon emergency detection, and finally, a class that constantly performs the conversion and inspection of the engine data (Figure 81).

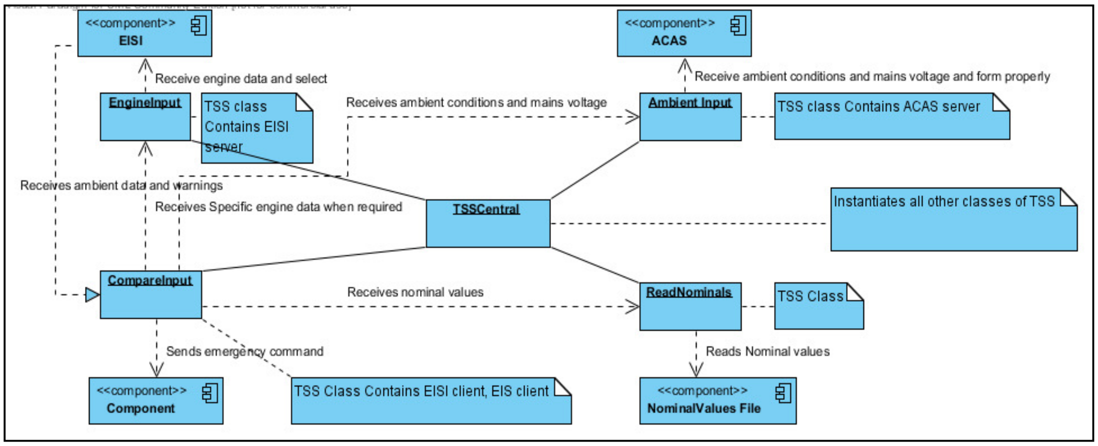


Figure 81 – Context model of the TSS

When the mainTSS() static function of the class TSSCentral() is invoked, the threads are instantiated and the Exchanger objects are passed in order to have the necessary data available in all 3 threads. The class CompareInput(), which conducts the correction and comparison of the critical parameters, has all the Exchangers from the other threads passed to it as well hence has all the available data whenever required (Figure 82).

The run() method of this class contains a while loop which constantly obtains data from the engine and ambient input threads. The class initially has the boolean compareMode attribute initialised to false. After that, the loop determines if the program has entered a compare mode. The check is done periodically every 10 seconds and it is based on the throttle position. If the throttle setting hasn't changed from the last position then the compareMode attribute is set to true. However, in order to encounter the extreme case of a rapid throttle movement ad repositioning at the previous setting, the throttle values are stored in an array after each loop and every 10 seconds the average value of the array elements is inspected. If the average converges to the previous throttle setting value, only then the program enters compare mode.

There are dedicated methods that check the relative exceedance of RPM, EGT, and also the absolute exceedance of RPM and EGT. The latter are checked continuously, even if the program is not in compare mode. If a relative exceedance is observed the corresponding indication illuminates on the user interface of the MEOCS. When the throttle is moved by the user, the caution message ceases to exist, until the program re-enters compare mode and performs the checks again. If a relative exceedance caution persists for 3 minutes without any command from the user, the AutomaticEngineShutdown() class is supplied with an alarm boolean and commences the safe shutdown of the engine and the program.

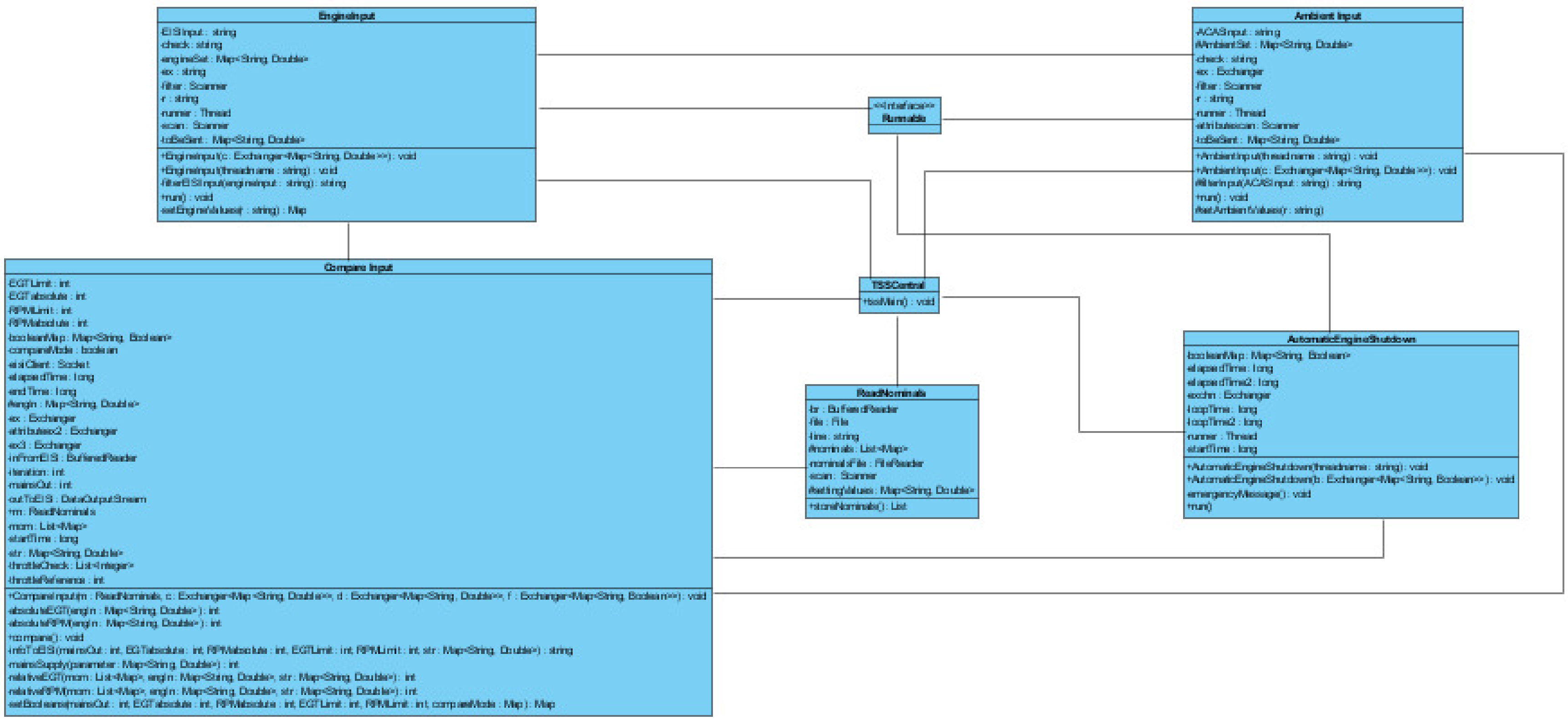


Figure 82 – UML class diagram of the TSS

Upon an absolute exceedance, the user is simply notified with a warning, and the automatic shutdown is informed appropriately to start the shutdown sequence. The mains power supply is also monitored continuously, even if the TSS is not in compare mode. The system allows 3 minutes to the user after power loss detection. If no action is taken, the AutomaticEngineShutdown() class is informed appropriately to start the shutdown sequence. The individual parameter check methods for relative exceedance are supplied with the nominal values List and each time they calculate the theta (θ) and delta (δ) parameters, based on the real time values of ambient temperature and pressure, and correct the received engine readings to ISASL conditions. After the completion of the checks, the ambient and peripherals information is transmitted via TCP to the EISI and consequently to the MEOCS to be displayed on the user interface.

Finally, the TSS was inspected for memory leaks and with all the modules integrated, the system was inspected for CPU usage and memory consumption: firstly with all the components installed on the same PC and secondly with the MEOCS installed separately.

7.4 Validation

Unit testing was applied to as many designed methods as possible. Methods with explicit return values were tested with JUnit generated test harnesses. The LabVIEW functions were tested with LabVIEW UTF where possible. The functions that produced physical signals were measured with the use of an oscilloscope.

Methods which required TCP connection were unit tested with the use of self-designed test harnesses: one acting as a TCP client and another as a TCP server. There were also several methods that were only possible to be tested during integration of the methods and the engagement of the General Test Harness, due to dependencies on other elements. Such methods were main functions, those attaching shutdown hooks or void methods writing to text fields and areas. The constructors were tested by observation of the instantiated objects. The unit test coverage of the involved modules is shown in Table 11.

Table 11 – Unit testing of designed modules

Module	Total methods	LabVIEW UTF	JUnit	Self-designed test harnesses	Integration tested
EIS	6	5	N/A	1	0
ACAS	4	2	N/A	1	2
TSS	18	N/A	12	1	5

After integration of the methods of the TSS, control flow testing followed, based on the test cases defined with the assistance of the estimated cyclomatic complexity (VG). This was derived from appropriate control flow diagrams (Appendix C, Section C.2.3). The integrated control flow diagram of the TSS (Figure C-9) was relatively complicated with a high cyclomatic complexity. However, it was possible to reduce it to three simplified sections with cyclomatic complexity values less than 10 and perform successful testing for each section (Figure C-9, Figure C-10). With regards to the Introduced LabVIEW functions, 2 of them contained a more complicated logic and required to be validated also by control flow testing (Figure C-5, Figure C-6).

Consequently, the integration testing followed, where the EISI, the MEOCS and the TSS were tested with the engagement of the General Test Harness (GTH) and all the previous control flow testing was repeated. Additionally, all the possible messages were recreated in the GTH and were successfully captured from the TSS and transmitted to the MEOCS user interface. The final stage of the validation of this round took place with the integration of all the LabVIEW components and the engine was operated as in the previous rounds. Special care was taken to perform boundary testing at the extreme throttle settings (0 and 100 %) and also at the absolute RPM and EGT limits definition.

7.5 Results – Discussion

The objectives of this round were accomplished with satisfactory results. The live camera image was captured and transmitted successfully, although the JNI call to the openCV native functions occasionally may have caused the JVM to crash during shutdown, leaving the signal resetting VI running. Although the exact native functions exceptions were able to be identified from the error logs of the JVM, no specific cause was able to be determined. Hence a safe and feasible solution to this problem would be to implement an additional LabVIEW function in this VI that would contain an invoke node and a Close Reference Express VI to shutdown if remained open. Additionally, the restarting of the RMI Server could be initiated with an external Java program, thus even if the current JVM instance has experienced an exception, the RMI Server will restart in a new JVM instance. The latter suggestion was implemented at the next round of the software process.

The TSS was completed and all the required functionality was implemented. It was successfully integrated with the other existing and newly designed components. There was a high percentage of unit testing of the implemented methods (Table 12) and the control flow testing of the various modules achieved 100% statement and decision coverage during validation (Appendix C, Sections C.1.5, C.1.6 and C.2.3). The cyclomatic complexity of the modules was maintained in satisfactory levels, allowing for adequate testing (Table 12).

Table 12 – Cyclomatic complexity of introduced modules

Module	Cyclomatic complexity
TSS part A	6
TSS part B	5
TSS mains supply check	3
EISI control status configuration	4
MSIS	3

The operation of the TSS was based on the derived nominal values of RPM and EGT. In future development, the CompareInput() class of TSS can be modified and adhere to an Abstract Factory Pattern, which will enable even more parameters to be monitored as critical, in the case of inclusion of different types of gas turbines. In this case, the class will be based on a standard interface (similarly as in EISI) and the concrete implementation will depend on the specific engine requirements. Currently the nominal values of the critical parameters were established after 13 samplings from actual engine runs. As it was a long process and the engine had a low MTBO, the project had to rely on this small sample. However, the sample was proven satisfactory, as no unnecessary or false warnings were created during the real engine runs. The validation of the TSS for detection of parameter exceedance was obtained through the GTH and successfully covered all the possible outcomes of the system. The corrected values of the statistical sample followed the expected trend throughout the operating envelope of the Olympus gas turbine (Figure 83, Figure 84).

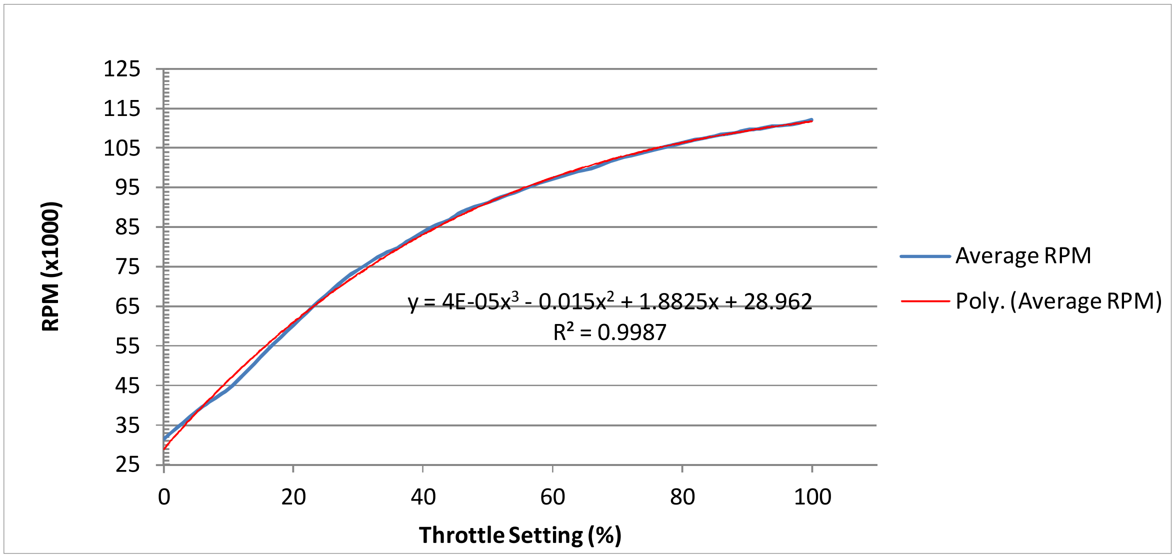


Figure 83 – Trend of RPM nominal values

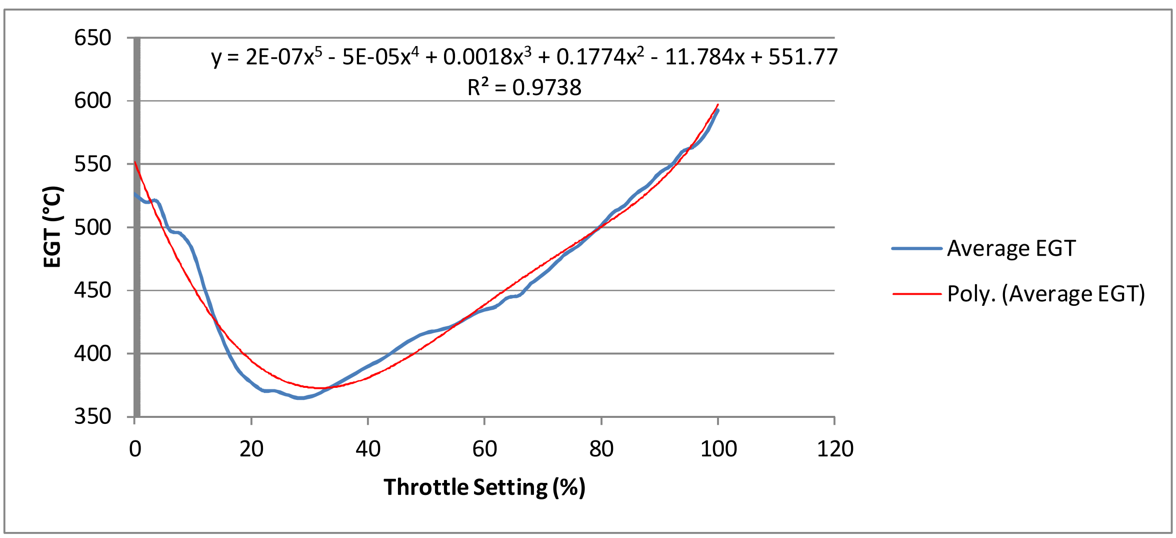


Figure 84 – Trend of EGT nominal values

The fuel flow signal was captured but as no reference values were available, great accuracy was not achieved. However, fuel flow was not regarded as a critical parameter and the main objective at this stage was to demonstrate the capability of capturing the signal and transmitting it through the network, rather than accurate representation of the fuel flow. Improved accuracy could be achieved by calibration of the transducer after completion of this project. All the other functions applied, operated as expected.

The TSS was investigated for potential memory leaks and revealed a stable operation with a low number of surviving generations that stabilised within the first 5 minutes of operation (Figure 85). Accordingly the heap size and usage were maintained at low levels with a stabilising trend (Figure 86).

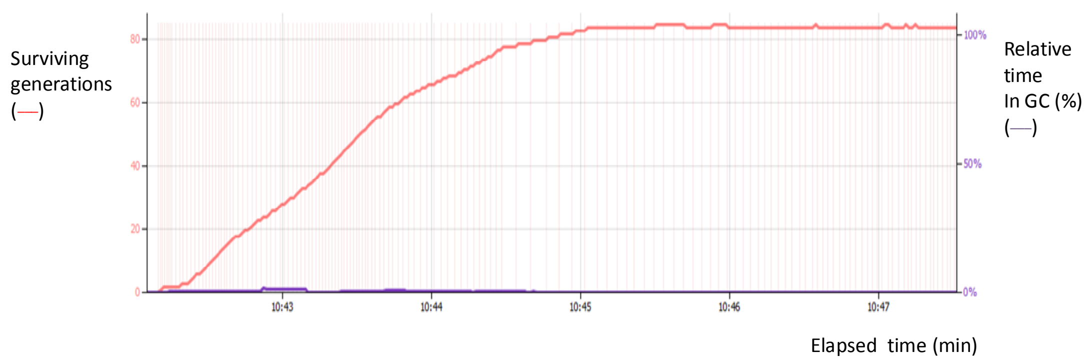


Figure 85 – TSS surviving generations

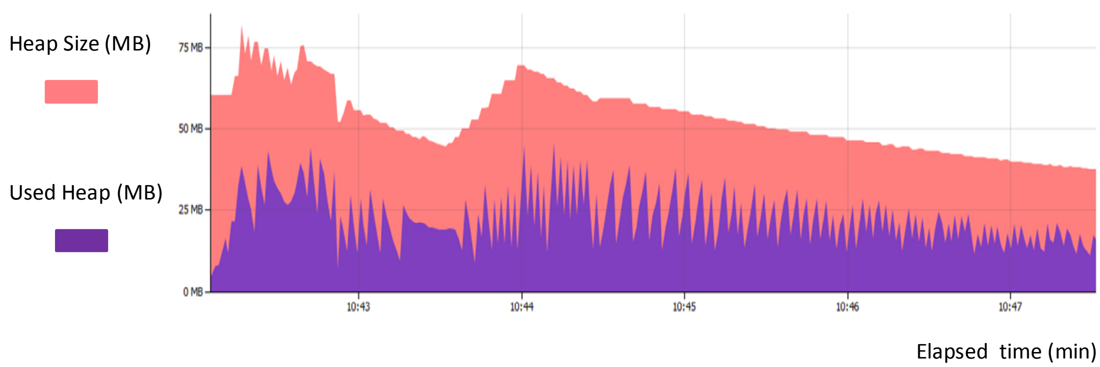


Figure 86 – TSS Heap size and usage

The MEOCS and the EISI were re-examined for memory leaks at this round again, as interaction with the TSS was added along with live camera image streaming. The MEOCS without camera input engaged, presented an increasing trend during the first 20 minutes until a sufficient garbage collection was performed by the JVM. From there on the operation was firmly stable with a very low population of surviving generations (Figure 87) and a heap size around 100 MB (Figure 88). The concentration of surviving object generations was caused mainly from increasing numbers of char [ ] objects, due to calls to Swing objects and Scanner objects. The former were involved with the user interface constant update and the latter with the mapping of the incoming information, in order for them to be recognized from the appropriate fields of the user interface.

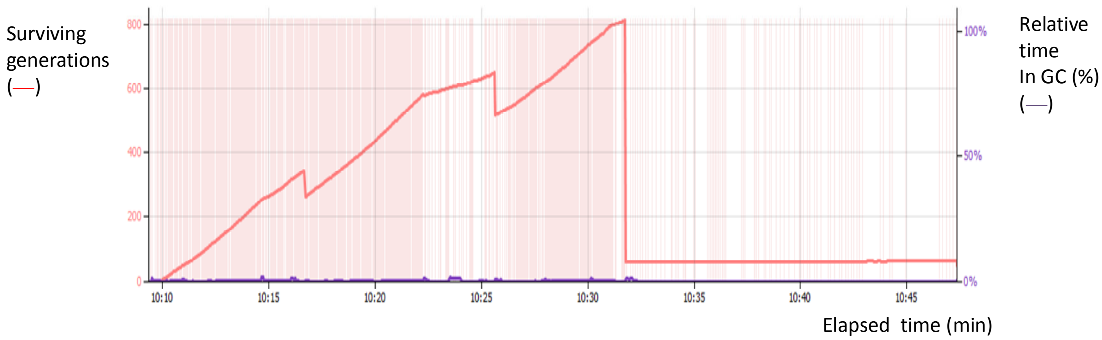


Figure 87 - MEOCS surviving generations – without live camera image

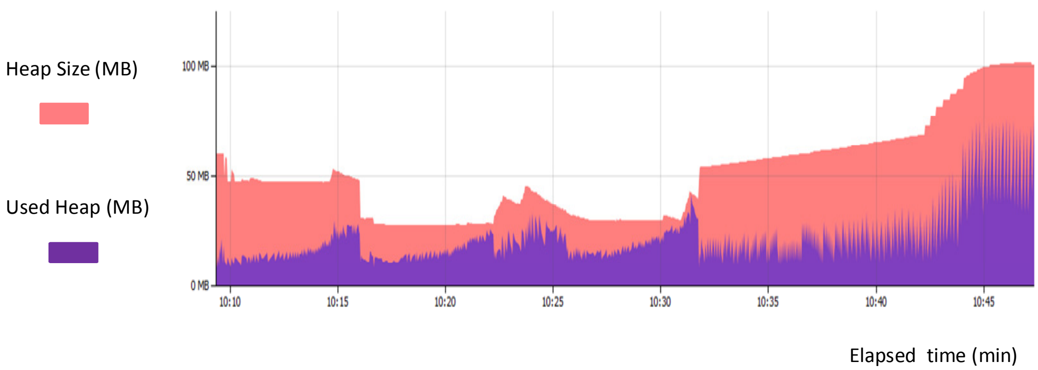


Figure 88 – MEOCS Heap size and usage – without live camera image

With the live camera stream engaged, the system appeared to stabilise earlier with around 60 surviving generations (Figure 89) and a noticeably higher heap (Figure 90), due to the continuous reception of byte arrays to display the image. In this case, byte [] objects were those with the highest surviving generations. However, it has been concluded that this high number has triggered the garbage collection earlier than in the previous configuration without the camera image, as there was a greater rate of unused objects concentration, resulting to earlier stabilisation of the operation.

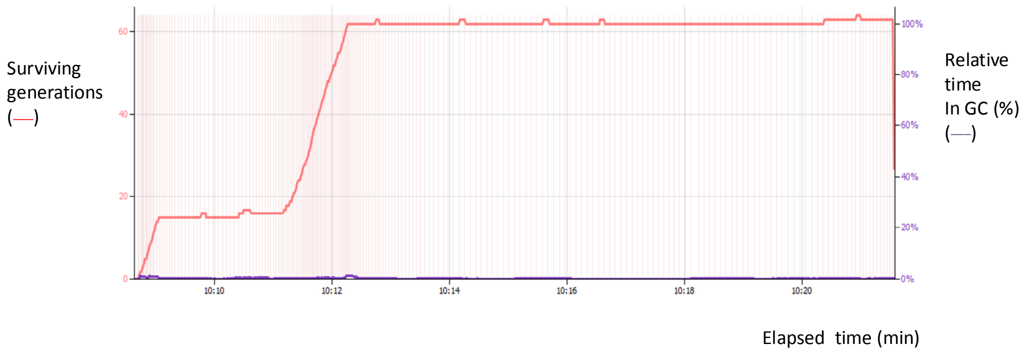


Figure 89 - MEOCS surviving generations – with live camera image

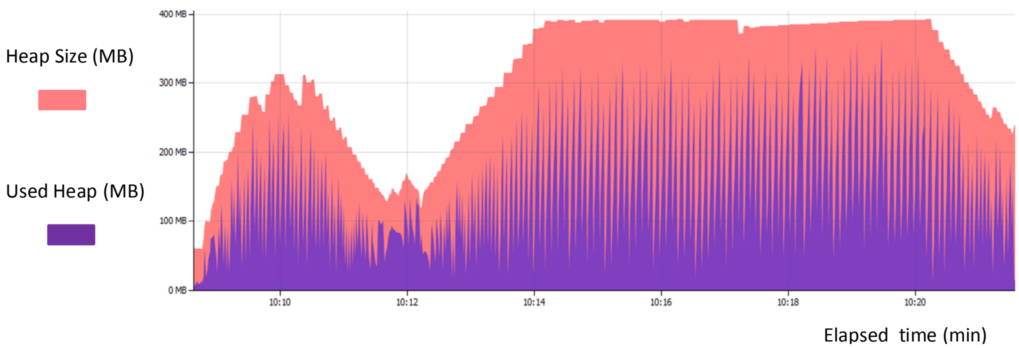


Figure 90 – MEOCS Heap size and usage – with live camera image

Similar observations were obtained from the EISI memory leak analysis. Without the camera engaged, the stabilisation occurred after nearly 25 minutes before continuation at a constant number of around 40 surviving generations (Figure 92) and a heap size of just above 50 MB (Figure 91).

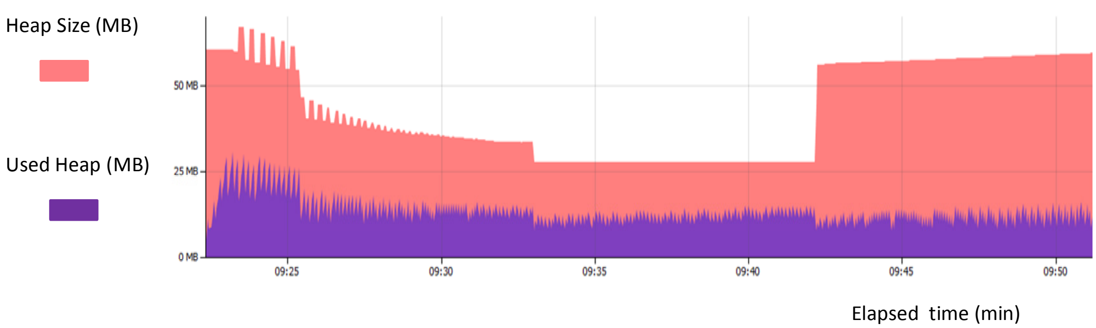


Figure 91 – EISI Heap size and usage – without live camera image

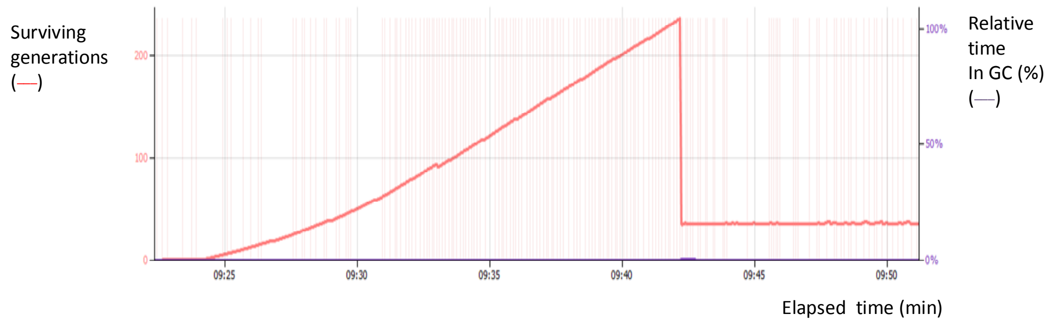


Figure 92 - EISI surviving generations – without live camera image

As with the MEOCS, the EISI with the live image thread engaged, stabilised much earlier (less than 10 minutes) than the configuration without the image (Figure 93). The heap size was accordingly stable, albeit a noticeably higher value (around 400 MB) (Figure 94).

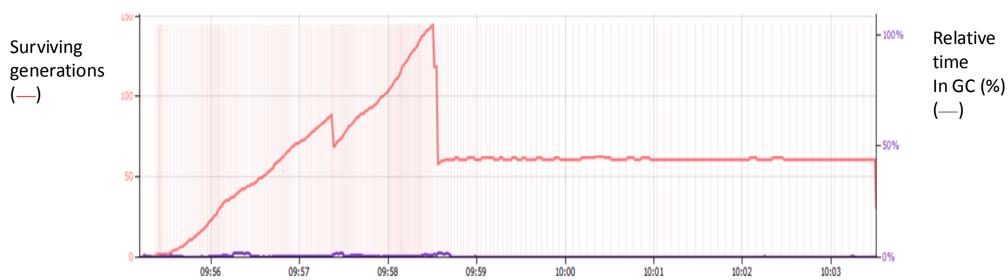


Figure 93 - EISI surviving generations – with live camera image

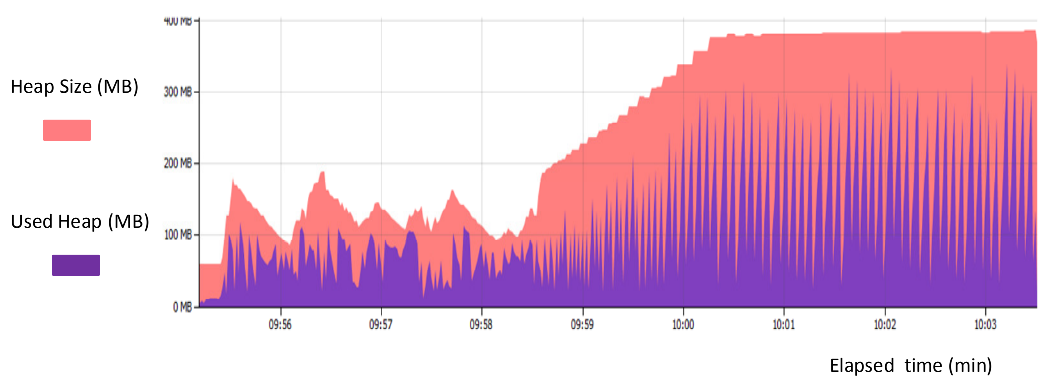


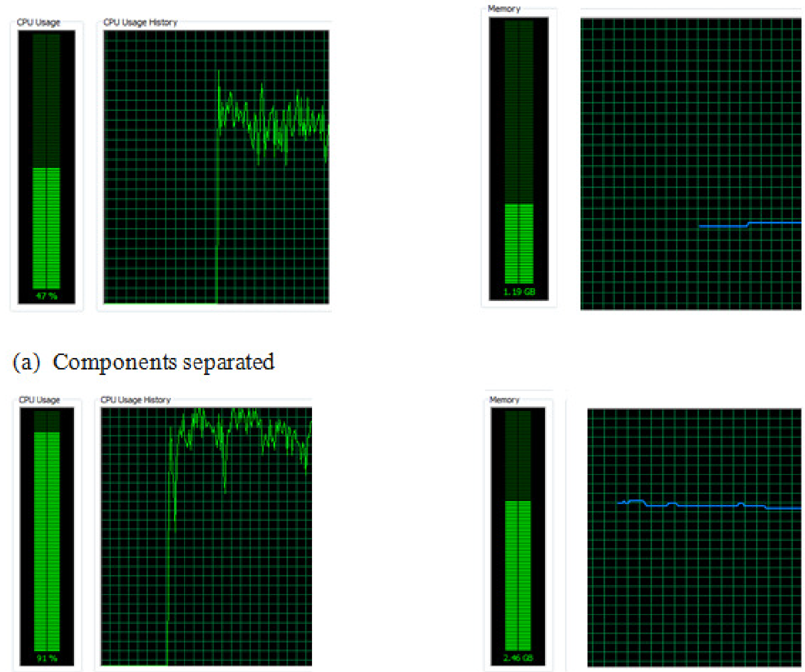
Figure 94 - EISI Heap size and usage – with live camera image

The 3 Java SE modules were also analysed with respect to quality metrics, in order to investigate clarity, maintainability and reusability. The separation in individual specialised software components has prevented excessive number of Lines of Code (LOC) in one program. The core modules, EISI and MEOCS, had an abstractness of 0.54, indicating good extensibility and reusability. An average instability (I) around 0.25 was also observed in the EISI and the MEOCS, and 0.0 in TSS, implying encapsulation, reusability and maintainability. It can also be seen that Lack of Cohesion of Methods (LOCM) was maintained relatively low, in an attempt to achieve cohesion in the designed classes. Avoidance of complexity was also obtained, indicating understanding of the applied algorithm. Evidence of low complexity - this is related to nested execution and element granularity and hierarchy - is the low average values of Cyclomatic complexity (VG), Nested Block Depth (NBD) and Weighted Methods per Class (WMC) (Table 13).

Table 13 – Code quality metrics

	LOC	NCP	A	I	LOCM	NBD	VG	WMC
EISI	1172	5	0.55	0.25	0.33	0	1	93
TSS	963	6	0	0	0.6	2	2	70
MEOCS	1243	4	0.5	0.25	0.32	0	1	61

According to the defined architecture, the EISI and the TSS must be installed in the same PC. The MEOCS however may be installed either together with the other 2 modules, or individually at a different computer within the LAN, without any modifications. The CPU usage when MEOCS and EISI were located separately was significantly reduced and maintained below 50% (Figure 95). The memory consumption was also significantly less. The option for separation is only available for the Internet version of the system, as discussed in the chapters to follow. For the LAN version, MEOCS must be installed separately.



(a) Components separated

(b) Components in same machine

Figure 95 - CPU usage and memory consumption on engine side PC

At this stage, a fully functional LAN prototype was completed (Figure 96). The user interface represents a complete operation panel of the Olympus gas turbine. The controls were situated on the left side of the panel, the main indications on the centre and the secondary engine indications along with ambient conditions were located on the right side. The caution panel was placed on the top of the panel whilst the emergency exit button on the bottom. The live image was displayed on the right top corner but can be moved around or resized by the user at anytime. The live image window cannot be closed on its own. It closes only when the whole panel is closed (Figure 97).

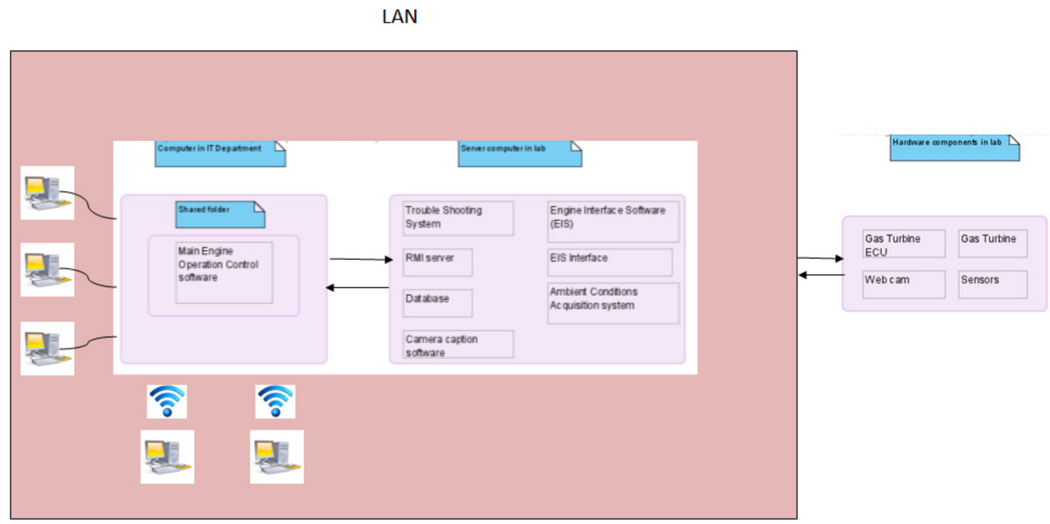


Figure 96 – Configuration of LAN version

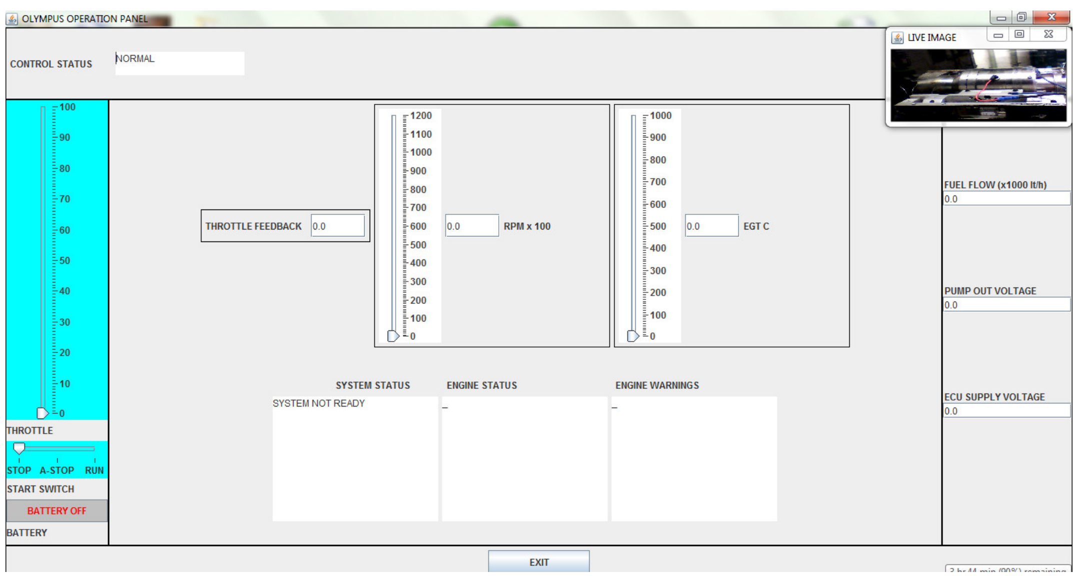


Figure 97 - LAN user Interface

The application was designed in a manner to ensure that only one user is allowed to run the engine at a time. This was obtained with the implementation of a hand shaking TCP connection between the RMI server and client. The connection remains open and awaiting for connection while the server is activated. It closes after connection with the client has been established and will re-open again after closure of the application, when the RMI server has been refreshed. The RMI refresh is an automated process. When the application is closed abruptly, the program ensures that the signals from the

EIS to the ECU have been set to values that enable the safe shut down of the gas turbine.

There are two ways to run the engine within the LAN:

- To have an installed MEOCS .jar executable file on any PC connected to the LAN (either wire or wireless)
- To have a MEOCS .jar executable file in a shared folder within the network

No modifications or adaptations of the MEOCS are required for either of the aforementioned options. Additionally, if the MEOCS is not installed locally with the EISI, it may be used as a switchboard to connect different gas turbines installed in different test houses, regardless if they are operating simultaneously. The prerequisites for this configuration are that each test house should have dedicated ports for RMI and data communication, and a dedicated operation panel. For the latter, the class which instantiates the panel, must be adapted to adhere to the Prototype pattern, where the required objects are created by using a prototypical instance and creating new objects by copying the prototype [49].

The physical risks (Appendix B, Section B.5.2) were sufficiently identified and the issue of supply power disruption has been significantly mitigated with the implementation of the TSS and the MSIS that monitor the mains power supply and take appropriate action when required. All of the hardware requirements were correctly identified and no complications or delays occurred due to wrong equipment procurement. Similarly, the software dependencies were satisfied and the validation was thorough and successful, indicating the problems discussed earlier in this section. Finally, the cost was maintained within the budget, as the equipment was purchased after careful market research and also by using existing components wherever that was possible.

8 INTERNET IMPLEMENTATION

This chapter describes the methodology and implementation of round 4 of the Spiral process. The main purpose of this round was to render the application accessible through the Internet. Basic Internet tools such as HTML, JavaScript, web services and AJAX were combined herein, in order to obtain a fully functional and reliable real – time remote application.

8.1 Methodology

The methodology applied in chapter 8 was guided from the objectives, shown in Table 14.

Table 14 – Objectives of round 4

Create a host website
Create a web application(s) to contain the application
Design appropriate API to invoke the methods of MEOCS
Design a safe and ergonomic user interface for web browsers
Secure the application
Deliver live video and/or sound stream through the Internet

8.1.1 Host website and web applications

There were several approaches considered for the implementation of the containing web application. These were narrowed down to PHP and Java EE frameworks. PHP is a scripting language for web development [138] and provides capabilities for dynamic web pages. The drawback of this option would be the requirement for installation of a Java bridge in several points to interface the Java SE methods of the MEOCS with PHP. A Java bridge is a Java EE application that requires a Java EE or Servlet to run [133]). Although this approach would exploit the capabilities of PHP, it would add serious complexity to the whole application, thus rendering it more prone to bugs and faults.

The other potential option was the Java EE frameworks. At this point two sub options were considered and discussed: The Java Servlet and Java Server Pages (JSP), and the JSF frameworks. The Java Servlet and Java Server Pages (JSP) frameworks are by default enabled when a Java EE web application is created in Netbeans IDE. According

to Oracle [115], Servlets are the Java platform technology of choice for extending and enhancing web servers. They provide a component-based, platform-independent method for building web-based applications, without the performance limitations of CGI programs. Also, Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases and can also access a library of HTTP-specific calls, receiving all the benefits of the mature Java language, including portability, performance, reusability, and crash protection.
JSP technology is an extension of the Servlet technology created to support authoring of HTML and XML pages and allow easier combination of fixed or static template data with dynamic content.

The second sub option if using Java EE, was the JavaServer Faces (JSF) framework. This is a server-side component framework for building Java technology-based web applications and consists of [111]:

- An API for representing components and managing their state; handling events, server-side validation and data conversion; defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features
- Tag libraries for adding components to web pages and for connecting components to server-side objects

From the developer's point of view, JSF consists of a collection of XHTML pages [153] and one or more JSF managed beans [66], which are lightweight container-managed objects (Plain-Old-Java-Objects, known as POJO) with minimal requirements. They support a small set of basic services, such as resource injection, lifecycle callbacks and interceptors [111]. Heffelfinger (2011) considers the JSF as 'a standard web application framework available in any Java EE compliant application server' but he does not imply that JSP, Servlet or other Java EE compatible frameworks are obsolete. JSF is regarded as a tool to provide easy steps for developers to create web pages, compared to the JSP and Servlet based Java EE frameworks.

Eventually, after trial attempts, which will be discussed in section 8.5 of this chapter, Java EE 6 with JSP 2.1 and Servlet technology as the supporting framework were selected for the development of the required web application. Glassfish 3.1.2.2 was chosen as the web server to contain the applications. This is the Java EE reference application server and it is by default embedded in the Netbeans IDE [66]. The Apache Tomcat [136] would be another option. Tomcat is a Servlet container that implements the part of Java EE that specifically relates to Servlets, thus additional parts of Java EE would need to be loaded from external libraries. Using Glassfish reduced the work load of setting up the application. The implementation of the various web pages was selected to be accomplished with HTML 4.0.1 [154], as some older browser editions do not support all the HTML 5 features [155].

8.1.2 Service invocation

After the rejection of the JSF option, the web service approach was studied, with two sub options available: WSDL web services or RESTful web services. WSDL web services (also known as XML web services) require an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards [143]. RESTful web services do not adhere to any particular web service protocols but represent an architectural design of services, abstracted from HTTP 1.1, as defined by Fielding (2000) [40]. It was discussed in chapter 3 of this Thesis that both approaches have many similarities, with REST proven to be more efficient with regards to flexibility and control. The WSDL services introduce dependency on vendors and open source projects [119]. As a general conclusion, RESTful services could be applied for tactical, ad hoc integration over the web and WSDL services to be preferred in professional enterprise application integration scenarios, with a longer lifespan and advanced QoS requirements.

In the current project, flexibility was required, as the perspective was to enable interaction with external applications in future development. Also, as continuous data exchange and requests would take place between the clients and the server, it would be beneficial to reduce the overhead required for the accomplishment of the transactions, resulting to reduction of the server load and response time. For these reasons, a RESTful approach would be appropriate. However, for increased security control, the WSDL approach would be more efficient. Hence it was concluded that a hybrid approach would be the best solution. Thus each one of the MEOCS static methods were mapped to a particular REST root resource that handles all the possible HTTP request types for that method only. Each root resource is a separate JAX – RS REST web service, where JAX – RS is the Java API for RESTful Web Services defined in JSR 311 [77]. Instead of having a fully RESTful API with each root resource contained in its own web application and the client in a separate one, they were all contained in the same web application allowing for enhanced security, which will be discussed at section 8.1.4. The application was not tightly committed to the proposed API, as at any stage it could easily be redefined in a more RESTful form and the security approach could be revised.

8.1.3 User Interface and live video transmission

The user interface was designed to be a practical and ergonomic gas turbine operation panel. It has followed the guidelines applied in the LAN prototype for separation of functions in three major groups: controls on the left, indications on the centre (with the most critical on top) and cautions/warnings on the right side of the page. It was designed on a JSP page with the use of HTML 4.0.1 [154] and CSS. CSS (Cascading Styling Sheets) [156] is a style sheet language used to define how to display HTML pages. CSS can either be embedded in an HTML page or provided as an external file.

Functionality to the page was added with the implementation of a corresponding JavaScript file. JavaScript is a scripting language used widely in contemporary web pages in order to add functionality and dynamic features [157]. It appeared in Netscape browser in 1995, and has been adopted by the ECMA a standard association since 1997 described by the ECMA-262 standard [33] .The control buttons and slider were implemented with the addition of YUI Library components and widgets. YUI is a JavaScript and CSS library for building interactive web applications [159], accessible either online or from the application server, if downloaded and installed. The buttons were designed with the use of YUI button widget, a mixed-module approach to providing a simple-to-use button that normalizes and enhances the web browser's default button component [160]. The throttle slider was created with the use of a YUI slider widget, a user interface control that allows the user to adjust values in a finite range along a horizontal or vertical axis. Typically, the Slider widget is used in a web application as a visual replacement for an input box that takes a number as input [161]. The indications of RPM and EGT were designed to be displayed on gauges, implemented with the use of gauge widgets from the Google Code Playground API's [58], which are a section of the Google Charts [57]. The Google Code Playground API's are also JavaScript and CSS libraries available online.

The acquisition of the engine and system output data from the browser operation panel, and the commands transmission are obtained with the implementation of Ajax (Asynchronous JavaScript XML) requests [143]. Appropriate XMLHTTP requests are sent to the REST root methods in very short intervals (2 per second) asynchronously, and upon reception of the reply, when the page is in stable condition, the required fields are updated with the information, without refreshing the whole page.

The web version of the program differs from the LAN prototype in the way the live camera image is captured and transmitted. A media server was installed to transmit the image via RTMP (Real Time Messaging Protocol), which requires constant streaming of the image to be available. For this particular project WOWZA media server version 3.5 was engaged [158], due to ease of configuration and implementation, although it is not open source software. However, this can be replaced in the future with an open source media server, such as Red5 [59]. The operation panel has included an embedded flash player that receives the video stream and displays it. For this project Flowplayer was used, an open source JavaScript library player [44]. The player library has been downloaded and installed locally in the domain of the web application. Flowplayer is also available in HTML 5 version.

The user can access the application through a separate Java EE web application, where the engine specifications are listed and access to other related information is available. The host web application consists of HTML 4.0.1 pages and has been designed to include provisions for future expansion of the application to include recorded engine performance data logs and operation information of various gas turbines. Finally, in the main web application, a training page was also created, which contains all the elements of the operation panel. However, it is inactive and only provides detailed information about the function of every element on the panel.

8.1.4 Security of the application

The web application was configured in a modular manner. The lowest tiers of the architecture are not exposed directly to the Internet, as they remain situated behind the firewall of the University. An apache HTTP server has been configured as a Reverse Proxy [135] and installed as the front end to the web applications. Virtual hosts have been designed to map the requests and redirect to the web applications. The virtual servers can be installed in the De-Militarised Zone (DMZ) of the IT Department of the University and accessed through the VPN tunnel. The web applications can be deployed in the Glassfish HTTP container behind the firewall. The application implements the Secure Socket Layer (SSL) protocol for data exchange. This is a cryptographic protocol that provides encryption of the data flowing between the end parts. Additionally, the back end of the web application (glassfish side), implements a form-based user authentication. Although this method requires noticeable work to be accomplished during implementation, it provides encryption of the transmitted password when the page has been set up to use HTTPS (HTTP over SSL) [66], as done in this case.

The Apache reverse proxy was configured to rely on OpenSSL to provide the cryptography engine, by using the built-in mod_ssl module of the server. However, during the time of writing, OpenSSL had a known vulnerability to the CVE-2014-0160 bug (Heartbleed exploit) [137]. This is a serious weakness in the OpenSSL cryptographic software library, which allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet [148]. An alternative recommendation would be to configure the reverse proxy to provide cryptography with the use of Network Security Services (NSS). NSS is available under the Mozilla Public License and supports SSL V2 and 3 [99]. This would be achievable with the use of mod_nss, which is an SSL provider derived from the mod_ssl module for the Apache web server and uses the Network Security Services (NSS) libraries [38].

The configuration of the web applications with a front and back end also enables flexibility of the program, as the main HTTP container (Glassfish) can be hosted anywhere conveniently and secured, behind the firewall. Hence the application can be easily accessible for administrators, provided that it is installed in physically secured areas, preventing any potential insider attacks.

8.1.5 Validation methodology

The Validation process included unit testing of individual JavaScript functions (with QUnit), inspection of the code, and control flow testing of complete functions (both JavaScript and root resources integrated with the MEOCS). QUnit is a JavaScript unit testing framework, used by the jQuery, jQuery UI and jQuery Mobile projects. It is capable of testing any generic JavaScript code. QUnit was originally developed by John Resig as part of jQuery and in 2008 it received its own home, name and API documentation, allowing for generic JavaScript unit testing. It depended on jQuery unitl 2009, when it was modified to run completely standalone [79].

Wherever unit testing with QUnit was not feasible, the introduced functions were tested against the General Test Harness (GTH) with the HTTP container (Glassfish) running, by adding them incrementally on the developed web pages. Every new function was integrated after being successfully unit tested. Integration testing was accomplished after combination of the individual modules and with the engagement of the GTH. Control flow testing was applied during the integration testing, based on the derived control flow diagrams of the designed software components and the calculated cyclomatic complexity [147].

8.2 Risk Assessment

Risk assessment of this round was accomplished with HAZOP. Mainly procedural risks were considered, as this round has not introduced hardware that would imply the necessity for further physical risk assessment, apart from the instability of Internet connection. The power supply loss risk has been mitigated in the previous rounds hence it was not addressed again. The Internet security was a combined risk addressed at round 0 and security has been identified as one of the objectives of the current round. Hence Internet security was not readdressed in this round’s risk assessment (Appendix B, Section B.6).

The risk assessment addresses the clarity of the objectives and also the identification of software dependencies. Cost Analysis was always considered and the new element added in this round was the risk of disruption of Internet connection.

8.3 Analysis, Design and Implementation

This section describes the newly added components along with the adaptations to the existing ones. The first sub-section presents a proposal for the hosting website. The next section describes the necessary final adaptations made to the EISI and the MEOCS to enable their interaction with the web application and the root resources. Consequently, the main web application was presented along with the user operation interface, concluding with several other features added to the application.

8.3.1 The host website

A dedicated Java EE web application was created to accommodate the host website. It is a simple combination of HTML pages with general information about the whole Internet Gas Turbine application and specific information about the Olympus gas turbine, which is in use. The landing index.jsp page was configured as the welcome page with a navigation bar to link to the other pages. The user will be able to navigate and find operational and performance data for the Olympus. Although these features have not yet been installed in the application, the pages have been designed with provisions to accommodate them in the future. There was a page dedicated to the specification of the Olympus (Figure 98), and the user can also find links to the operation manual and the manufacturer's website. The same configuration will be maintained for other gas turbines to be added in the future (Figure 99).

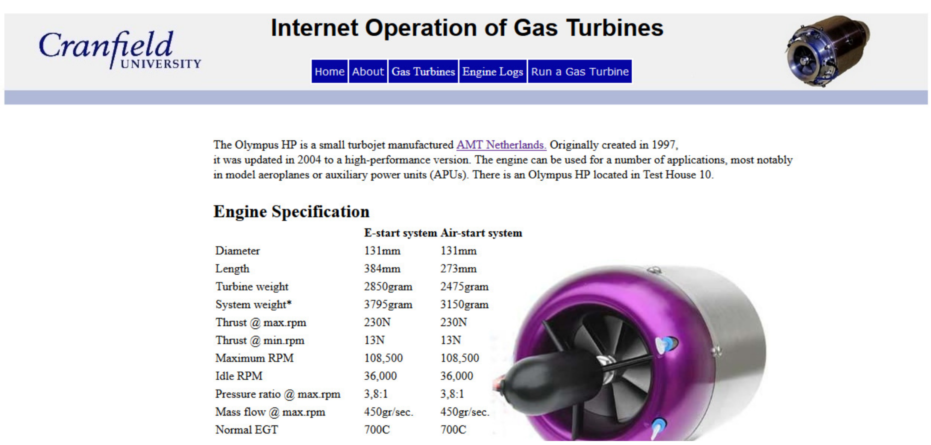


Figure 98 – Specification web page

From the page of Test House 10, the user can select to visit the Olympus operation application. If more gas turbines are added in the future in Test House 10, the user will be able to navigate to the corresponding application from this page. Similar hierarchy was proposed for other Test Houses to be connected to the project in the future.

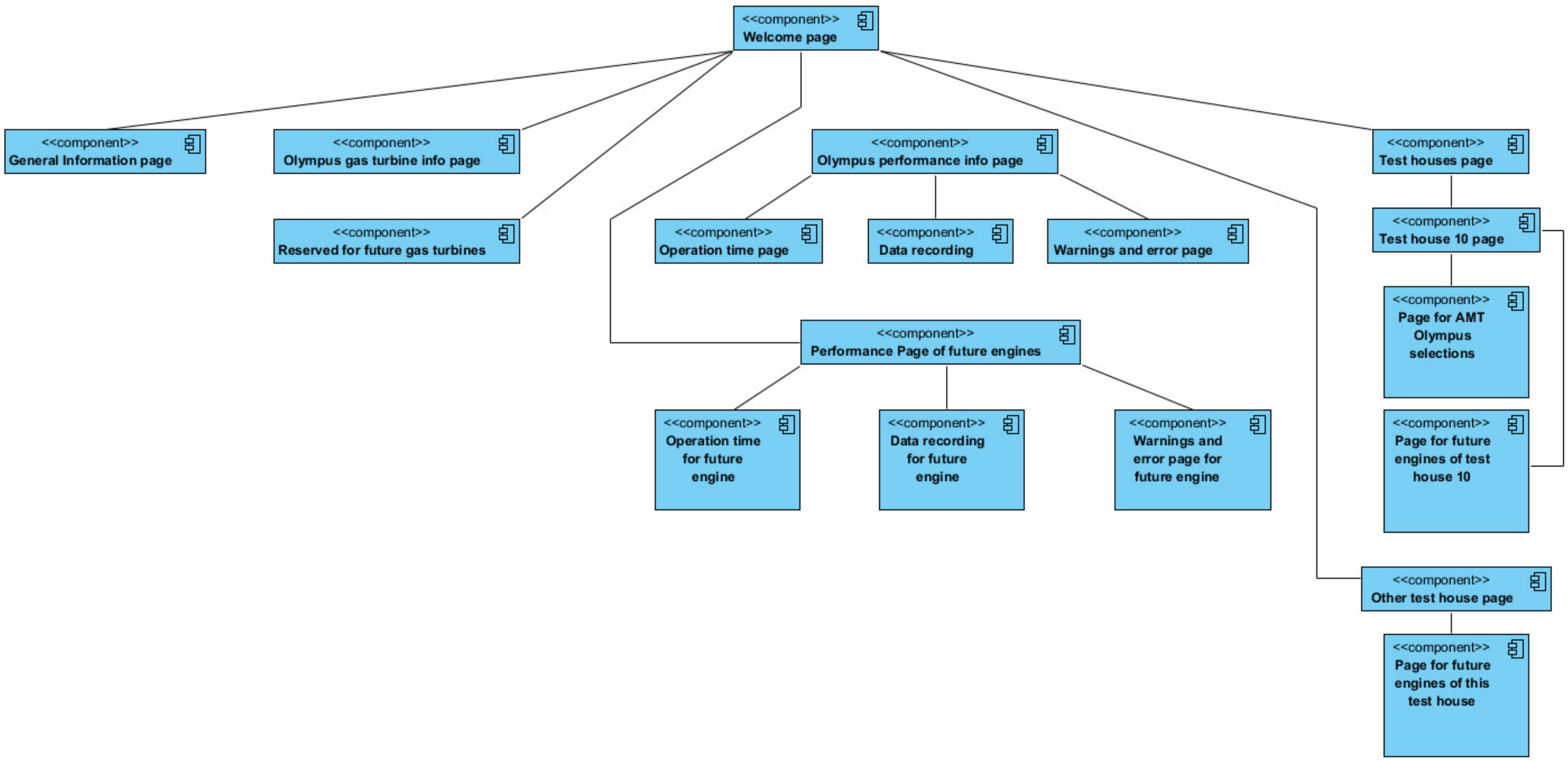


Figure 99 – Web pages of the host website

8.3.2 The main web application

The web application (InternetGasturbine) has been designed to incorporate a life cycle with 6 potential states, not all sequential. The initial state was assumed to be that when no activity is present, other than the servers ready to accept connection. The following state is the idle state, where the landing page has opened and the program awaits selection from the user to execute the engine operation application. The user reaches this state after he has navigated from the appropriate test house web page in the host website, provided that he has logged in successfully (Figure 100).

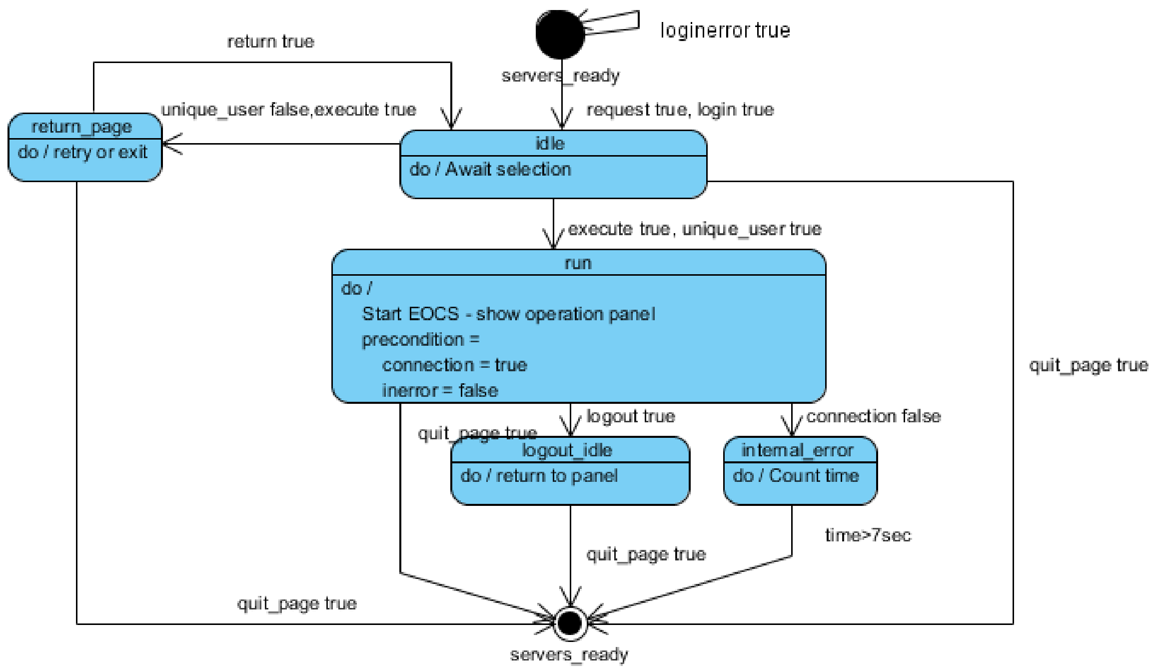


Figure 100 – State machine of the web application

If the existing state is idle, the user can then select to run the engine, hence proceed to state 'run'. The transition to this state would only be possible if the engine is not already in use from another operator. Following the 'run' state, should the program detect a connection problem, it must transit to the 'internal error' state, programmatically. Selection of exit request on any window will result to transition to the initial state, as with a logout request (Figure 100).

The web application hierarchy contains 5 main folders, from which: one contains the web pages, one the source packages, one the libraries, another the REST web services and finally one contains the generated configuration files. The web pages folder includes general pages and several structured folders for dedicated files (Flowplayer, JavaScript files, observer files, operator files, simple user files and training pages) (Figure 101).

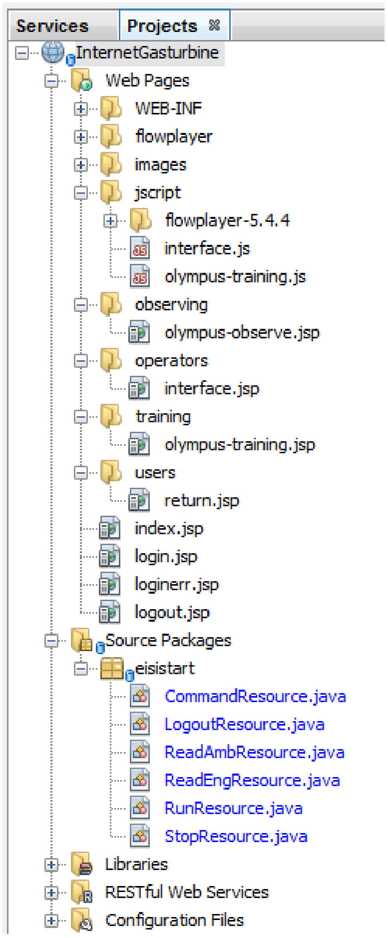


Figure 101 – Hierarchy of the main web application

The Apache reverse proxy was configured with virtual hosts to allow access of the application through HTTPS over SSL. With the Apache server as a front end, the application was then contained in Glassfish as the back end behind the firewall. The final architecture of the application comprised of 5 tiers, including the end remote user and the gas turbine with the related hardware (Figure 102). With the reverse proxy configuration, the proposal of this project was to install the Apache SSL virtual host in the DMZ of the IT Department, accessible through the VPN and the Glassfish container somewhere physically secured behind the firewall. The EISI, the TSS and the EIS modules are required to be installed in the test house computer, along with the camera and the media server.

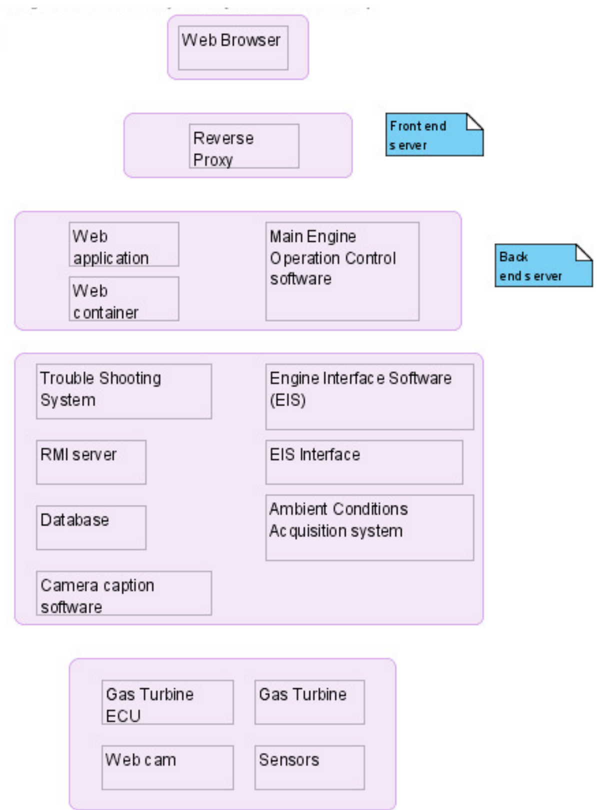


Figure 102 – Architecture of the integrated application

Apart from the SSL encryption and the suggestion for access through the VPN, the back end application has also been secured with a second layer of verification, through a form – based authentication by enabling a security realm.

8.3.3 Functionality and the REST API

The functionality of the main user interface web page was added by JavaScript methods and widgets. There were six YUI widgets required to create the controls of the engine and two Google API's widgets for the design of the RPM and EGT gauges. The controls create AJAX requests to send the commands to the resource. AJAX was also implemented to request data output from the server side, in order to allow update of the involved fields only, without triggering the refresh of the whole page. Hence two functions were required for these tasks and additionally two more to update the gauge widgets. Finally, a flash player function was embedded to receive the live video and sound stream (Figure 103).

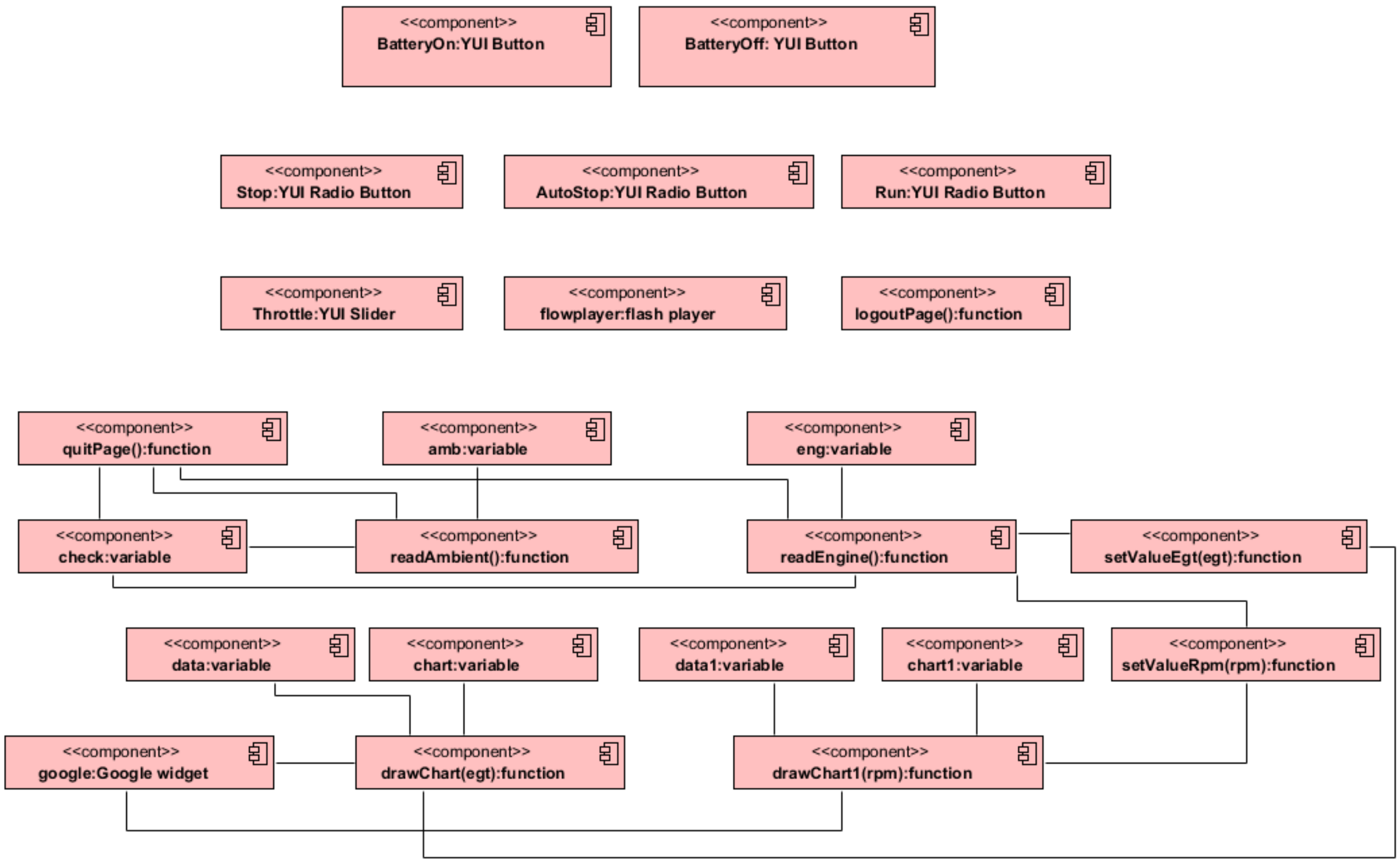


Figure 103 – Context diagram of the user interface JavaScript functions and widgets

For clarity, the functionality was included in a separate JavaScript file, called as an external script from the operation panel page. This file essentially was a JavaScript client for the root resources implemented to access the Java SE static methods of the MEOCS. In total, 6 individual requests are generated from the client during the lifecycle. Each request is directly mapped to a corresponding root resource (Figure 104), which consequently calls the appropriate MEOCS static method to either change a local resource or retrieve it. The requests that need to alter a resource are provided as POST requests and those that simply retrieve one are given as GET requests (Figure 105). The 2 data acquisition AJAX requests are transmitted programmatically every 500 ms, invoking the data acquisition process at the engine ECU, and the peripheral sensors and hardware components.

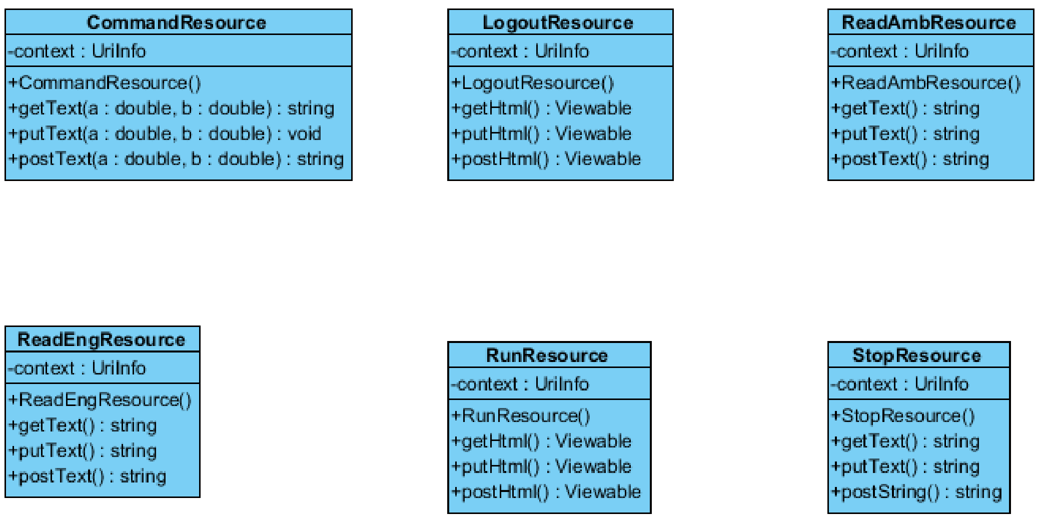


Figure 104 – UML class diagram of designed REST web services – No association between them

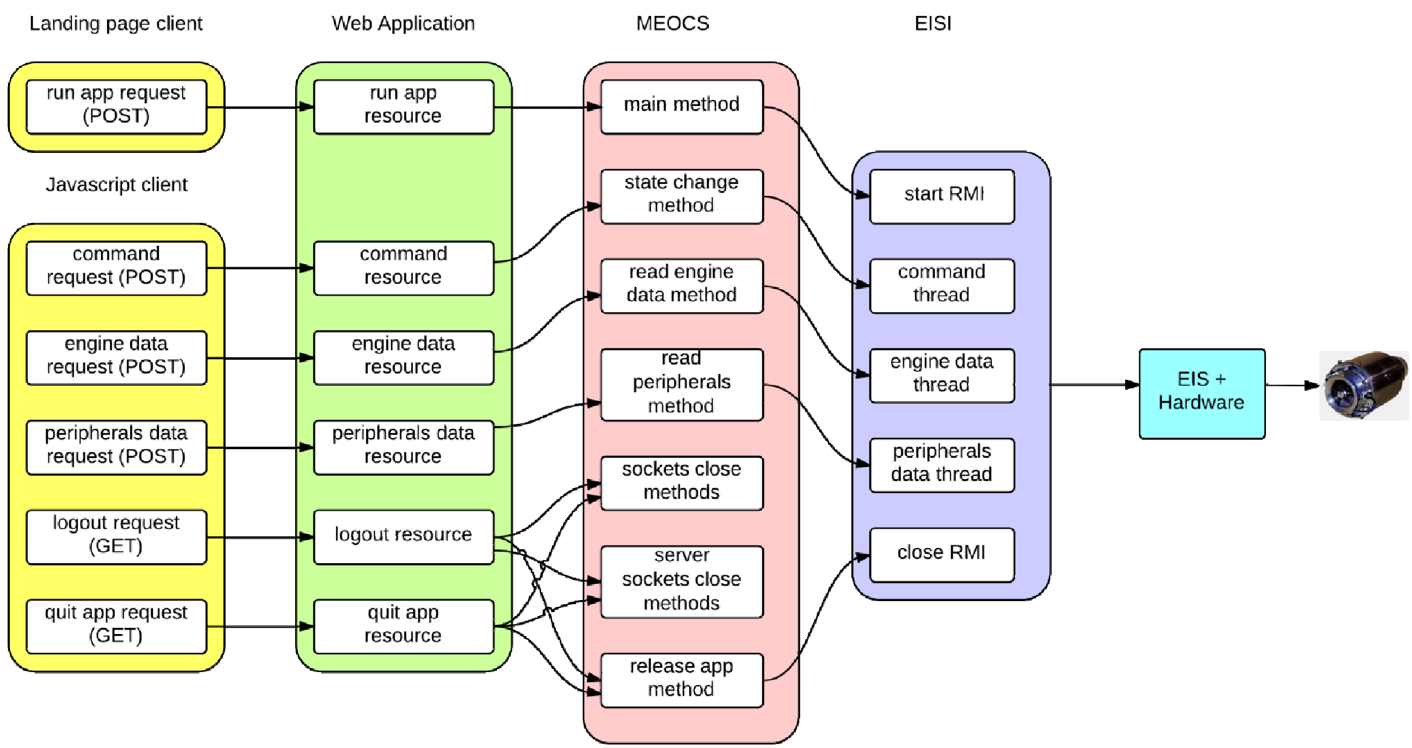


Figure 105 – Layout of the REST approach for resource retrieval

8.3.4 Adaptations of existing modules

The adaptations of the Engine Operating Control System (EOCS) were located mainly in the MEOCS and the EISI, and also a new monitoring program was added (Figure 106). The monitoring program upon start opens a TCP connection and transmits a handshake signal to the RMI server. If the server is not running, the socket exception in the monitoring program will trigger a restart of the RMI server and consequently – before shutting down programmatically – the already existing RestartServer application, which in turn will restart the monitoring program (Figure 107).

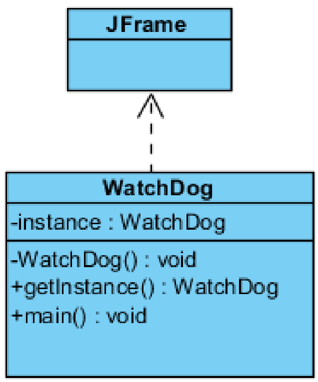


Figure 106 – UML class diagram of the monitoring program

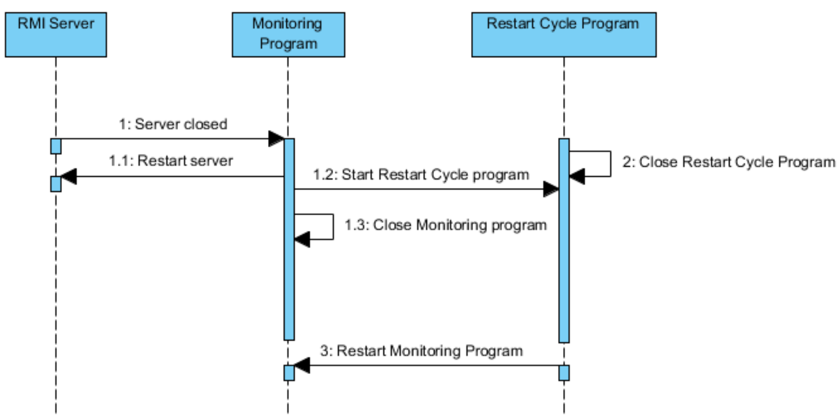


Figure 107 – Sequence diagram of the RMI server restarting process

There were not any serious modifications required for the EISI. A web version was created as a new branch in the repository. The main difference from the LAN version was the addition of 2 new classes and the disengagement of the Camera Capture Thread. The first class added (RestartInfo) runs on a dedicated thread upon start of the RMI server and provides a TCP communication port with the newly introduced monitoring program to indicate operation of the server. The second class added (RefreshServer) is instantiated upon the creation of the Registry instance when the RMI action (Figure 108) method is invoked. This class runs a thread with a TCP channel awaiting a closure command from the MEOCS. Upon reception of the closure command it unbinds the RMI Registry and initiates the shut down sequence (Figure 108). Finally, a method for checking the throttle command for rapid input (throttleCheck) was added in the ClientEIS() class, after being exhaustively unit tested with JUnit. This feature was removed from the MEOCS where it was installed in the LAN prototype.

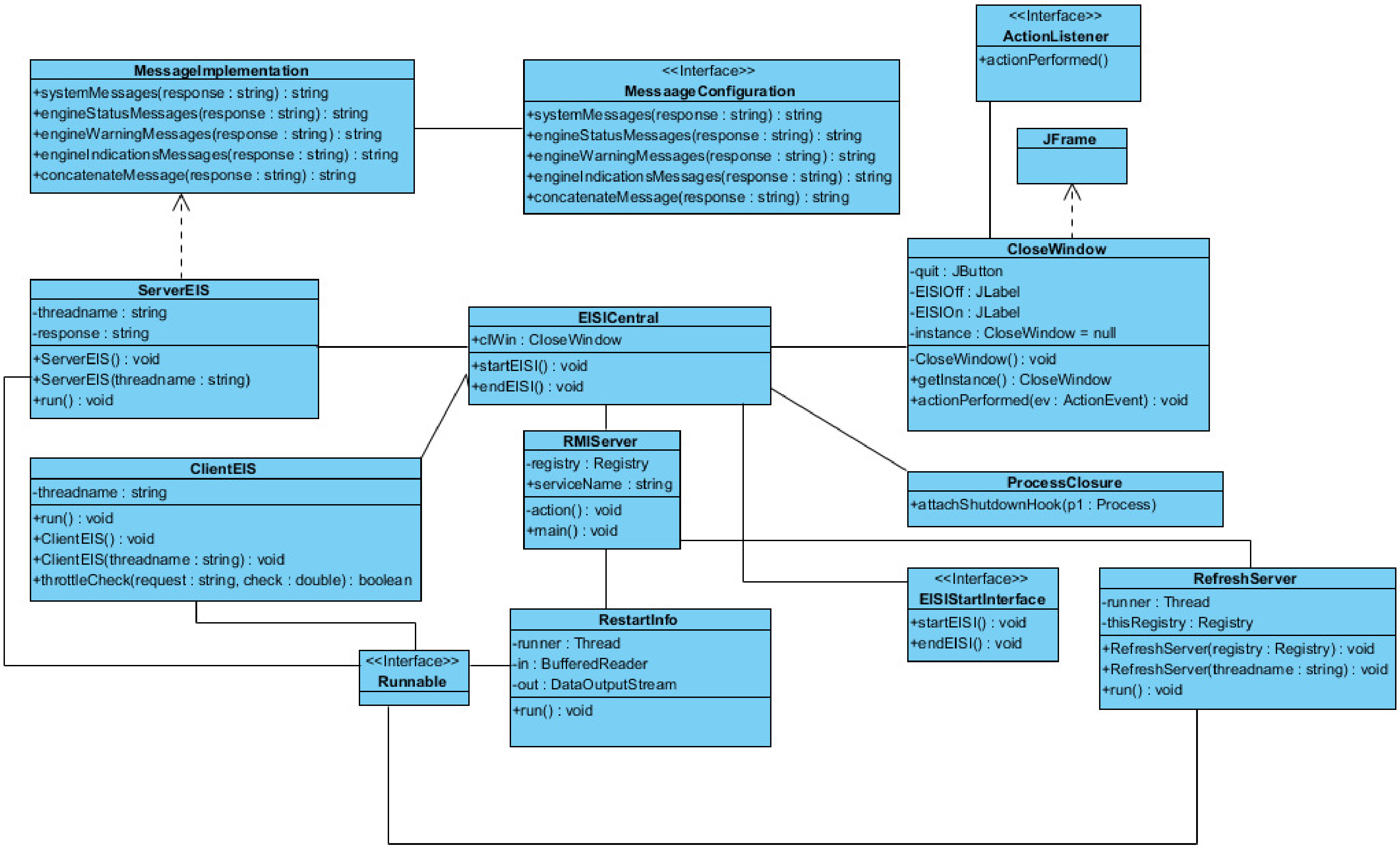


Figure 108 – UML class diagram of EISI web version

The MEOCS module had to undergo the most noticeable modifications. The Java GUI was disengaged and the throttle checking function was moved to the EISI. Instead of triggering the transmission of a command as an action event, the CommandSender() class now performs this task after the startCommand() static method (Figure 109) is called from the corresponding root resource. As for data reception, the sent TCP package remains in readiness until the two appropriate static methods in classes TSSInfo() and MessageReceiver() are called from their corresponding root resources (Figure 109). The response to the respective TCP clients in the EISI triggers the sequence for the transmission of a new set of data output. The difference from the LAN prototype in these tasks is that the data is not constantly received inside while loops.

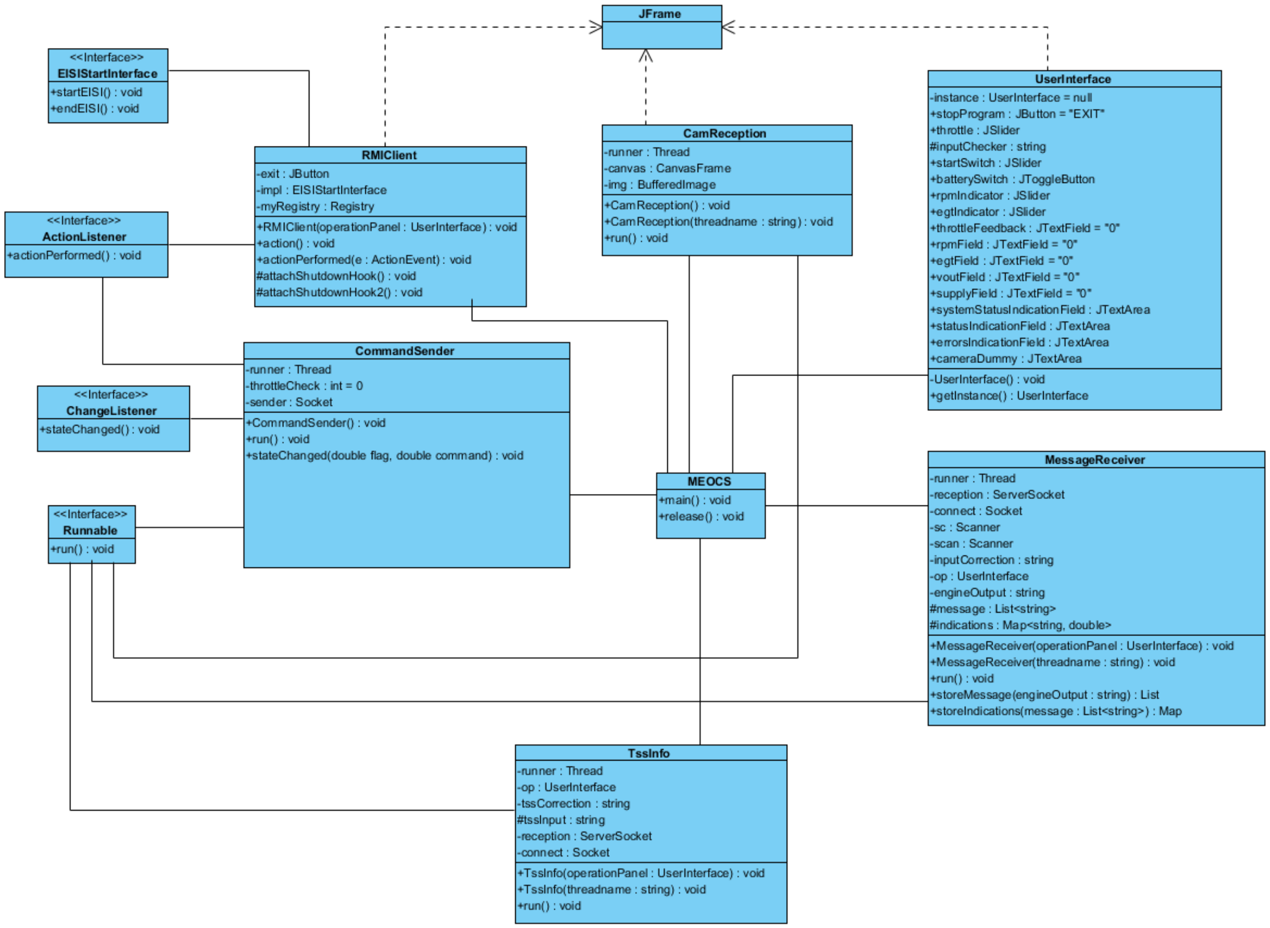


Figure 109 – UML class diagram of the MEOCS web version

8.3.5 The operation panel web page

The operation panel page was designed as the main interaction interface of the remote user with the server end. It was designed ergonomically with control, indications and warnings placed in 3 separate columns (Figure 110). The page was designed to allow the panel to be totally visible in any type of screen, without the need for the user to scroll down. RPM and EGT indications - as the most critical parameters - have been placed on top of the indications column and they are presented both in digital and gauge form, allowing the user to visualize the permissible operating range of the parameters and prevent limits exceedance. The reception of a command on the server side is acknowledged by the illumination of a green indication underneath the corresponding control widget. For the throttle, if a new setting greater than 20 % from the previous is selected, the command will be rejected at the server side and the response will trigger the illumination of a red indication light under the throttle.

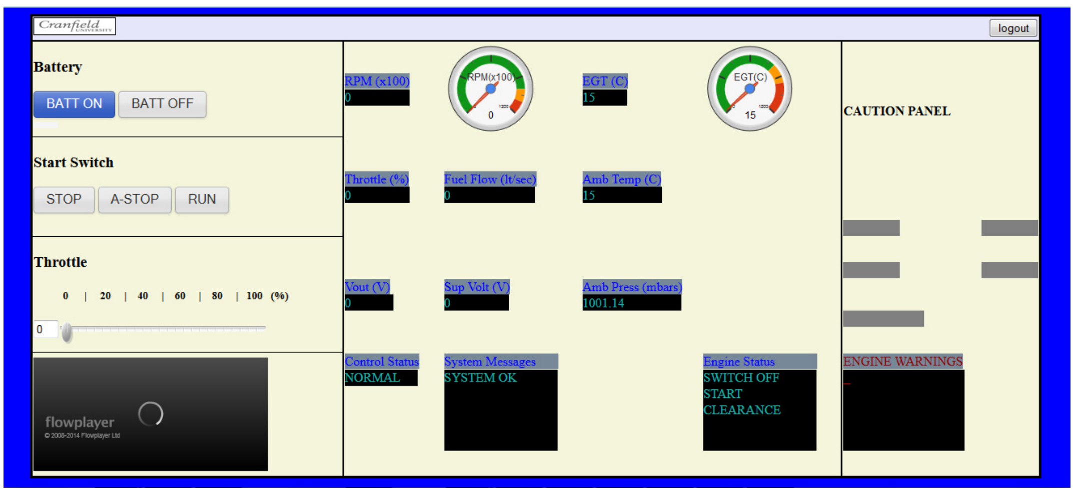


Figure 110 – Operation panel web page

8.3.6 Additional Functionality

An important feature that was added in the application was the ability to commence the shutdown sequence if there is a disruption or delay in the Internet connection. This task is triggered after timing out of the server sockets in the EISI threads for data reception, at the engine side. As described in the previous section, the response from the MEOCS server sockets triggers the sequence for the acquisition of a new set of data. Hence a delay in the arrival of the response (greater than 7 sec) triggers the timeout exception of the sockets, followed by the system shutdown. The timeout checks take place in 2 threads of the EISI (Figure 108). The expiration time of the sockets was selected without any particular guidelines other than high sensitivity required for the system to prevent unattended operation of the gas turbine. A lower expiration time would render the system extremely sensitive causing problems in the start up sequence, and time greater than 10 sec was regarded as insufficiently long.

The web application has been configured from the web.xml file to timeout in 5 minutes rather than 30, which was the default setting for a Glassfish web application. Additionally, if there is no command for 3 minutes, the corresponding socket in ClientEIS() class of the EISI will time out, preventing the user from leaving the application to run unattended. Every time a server side closure exception or shutdown is triggered, the shutdown hooks in the EISI cater for the safe shutdown of the engine and reset of the system. The browser operation panel is hidden from the user and the AJAX requests of the page are disabled, in order to prevent non deliberate DoS to the remote ports engaged. These actions also take place if the browser JavaScript client detects that the received data are not in the expected form, or if the XMLHTTP response status is other than 200.

Finally the live video stream was successfully captured and transmitted with the WOWZA media server to the embedded flash player of the operation panel web page. The video is also accompanied by sound and it is captured in a simple chat mode of the media server, through an ordinary web cam. The RTMP transmission was also proxied through the Apache reverse proxy and transmitted to the client through an HTTP virtual host. There has been no interference with the SSL port used for the main application.

8.4 Validation

Unit testing was applied to as many designed methods as possible. Java SE methods with explicit return values that were newly added were tested with JUnit generated test harnesses. The REST web services were individually assessed prior to integration by implementing simple JavaScript clients that sent the required requests. Finally, the created JavaScript functions were tested with the design of appropriate QUnit test harnesses. Methods which required TCP connection were unit tested with the use of self-designed test harness, specifically one acting as a TCP client and another as a TCP server. There were also several methods that were only possible to be tested during integration of the methods and the use of the General Test Harness, due to dependencies on other elements. The unit test coverage of the involved modules is shown in Table 15 (This table also includes the unit tests of the EISI and the MEOCS modules, which were accomplished at previous rounds).

Table 15 – Unit testing of designed modules

Module	Total methods	QUnit	JUnit	Self-designed test harnesses	Integration tested
EISI	26	N/A	9	5	12
MEOCS	22	N/A	3	5	14
REST services	18	N/A	0	18	0
Interface functionality	12	4	N/A	6	2

Control flow testing was then applied to the JavaScript client with the REST web services integrated in the web application. The testing commenced with YUI widgets added and tested incrementally, before adding and testing all the other JavaScript methods. After integration, all the control flow tests applicable to each JavaScript method were repeated. The test cases were defined with the estimation of the cyclomatic complexity, based on the derived control flow diagrams (Appendix C, Section C.3).

The testing was initially accomplished against the GTH. After successful integration, all possible warning and error scenarios were recreated with the use of the GTH, before performing actual runs of the gas turbine. The test runs of the gas turbine followed the guidelines of the previous rounds tests, with incremental increase of the throttle settings in intervals of 20%, reduction in a similar manner and also rapid reduction and increase.

8.5 Results, Discussion

The objectives of this round were successfully accomplished despite the initial approach problems of the web applications. A useful host website was created, which is a simple design without any complicated embedded functionality. All the required information are available there, along with referrals and links to the gas turbine manufacturer, and to Cranfield University. It was designed without unnecessary effects and the navigation bar that links to the other pages was located centrally with self explanatory buttons (Figure 111).

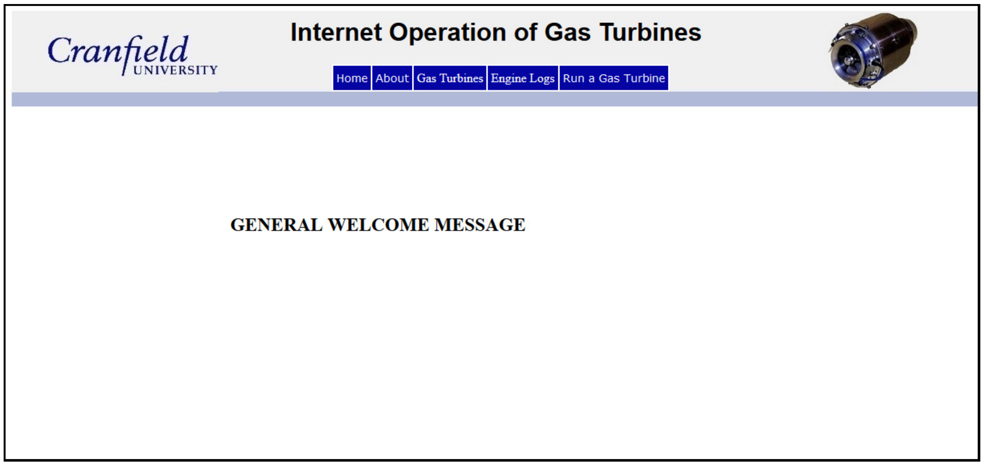


Figure 111 – Welcome page of host website

The web application that contains the MEOCS and the functionality was initially attempted to be designed with a JSF framework. JSF implements XHTML pages, which are usually recommended for presentation oriented web applications. The JSF facelets would be beneficial for the display of the operation panel web page, provided that the required functionality would be supported by the managed beans. At the early stages of the web application development, it was observed that it was not feasible to provide parameters and data to the managed beans in order to transfer them to the appropriate Java SE methods. For this reason, the approach was diverted to the design of a service oriented application with the JSP Servlet framework.

The REST style API was designed to accept the client requests and invoke the SE methods. It is flexible and reduces the overhead that would be required for a WSDL or Servlet approach. However, there is much functionality - such as data or users filtering – that could still be added with introduction of new Servlets in future development. The simplicity of the introduced REST web services will also allow the execution of the server side Java SE methods from other relevant external applications, as discussed in Chapter 10. Although in a more REST approach it is usually common to use GET requests rather than POST, it was observed in this case that for data acquisition in Internet Explorer, the response was returned from cache with status 304, preventing the supply of refreshed engine and peripheral data to the Internet user. In this project cache was not of interest. Although this might contradict the philosophy of the HTTP protocol, as addressed by Fielding, the application was designed not to operate with data from cache. For this reason, requests that alter the status of the server side Java SE resources were implemented as POST requests. An alternative to this would be to implement GET requests with an embedded JavaScript function to add a random number to the request URL, preventing response from cache. However, for simplicity, the POST approached was preferred.

The interface has functioned as expected during this primary stage of testing, although it has shown high sensitivity to connection disruptions and to erroneous data reception from the server side. However, for all the observed disruptions, the system has responded appropriately as expected and the server is always aware about the status of the client (running, closure or lost connection). At this stage, a licensed media server was used, do to ease of configuration. This feature increased the total cost of the application, but it was the only element that was added to the cost increase in this round. In chapter 11, alternative suggestions for open source media servers are discussed.

The configuration of the application to run over SSL enabled the capability of data encryption and in combination with the disabling of cache, the exchanged data is not easily available for potential hackers to obtain and decode, in order to perform a malicious command transfer to damage the gas turbine. The addition of the reverse proxy as a front end allows the main containing server to be positioned behind the firewall of the University, reducing the risk for port scanning and potential exploit attack or DoS (Figure 112 – Arrangement of front and back end servers). Moreover, the form based authentication of the Glassfish domain combined with the proposed access through the University VPN added a double level security authentication, preventing exposure to unauthorised users.

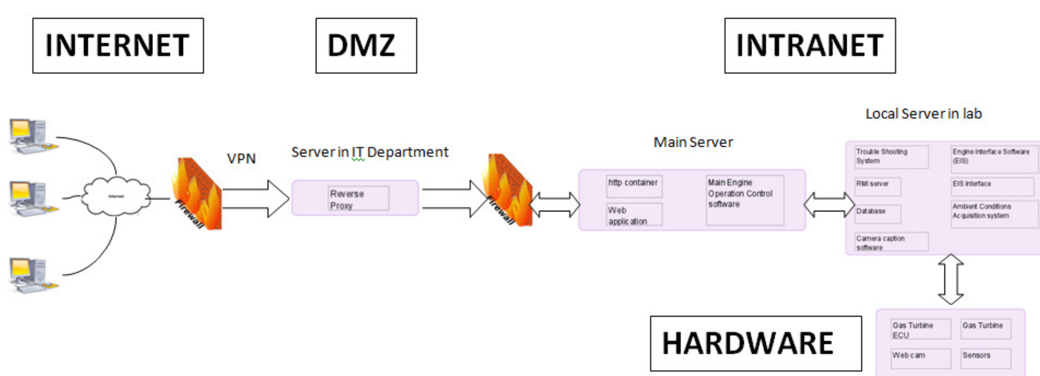


Figure 112 – Arrangement of front and back end servers

The training page that was designed was totally based on the operation panel, and the YUI and Google widgets. The functionality of the operation panel was omitted and images in the form of question marks were placed adjacently to all the page elements. It was based on the simple concept that should the user hovers the mouse over the question mark images, the corresponding title of that image appears with detailed explanation of the functions of that particular element (Figure 113).

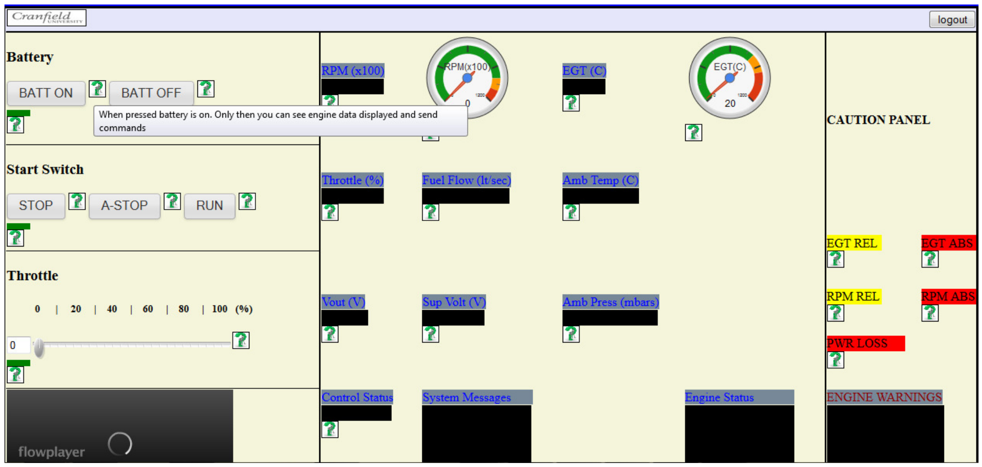


Figure 113 – Training web page

The validation of the web applications and the individual JavaScript functions was accomplished in 5 major web browsers and 3 different operating systems, in various editions (Table 16). There was a high percentage of unit testing of the implemented methods (Table 15 – Unit testing of designed modules) and the control flow testing of the various modules achieved 100% statement and decision coverage during validation (12.2C.3). The cyclomatic complexity of the JavaScript modules was maintained in low levels, allowing for adequate testing (Table 17).

Table 16 – Operating systems and web browsers used to test the application

Browsers							
Operating Systems	Microsoft Windows xp	Windows Vista	Windows 7	Microsoft Windows 8	Mac OS X	Linux	

Table 17 – Cyclomatic complexity of JavaScript methods

Module	Cyclomatic complexity
YUI button functions	5
YUI slider functions	5
Peripherals read function	4
Engine read function	3
Quit function	4
Logout function	3

The main web application (InternetGasturbine) was investigated for potential memory leaks and revealed a stable operation with a low number of surviving generations that stabilised within the first minutes of operation (Figure 114). Accordingly, the heap size and usage was maintained in low levels with a stabilising trend (Figure 115). The application was invoked several times and redeployed deliberately but maintained a stable population of few surviving generations of objects.

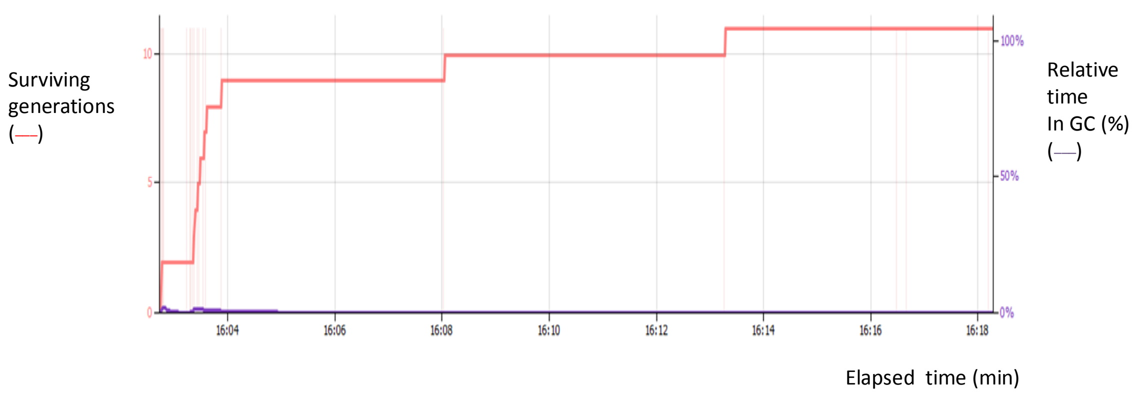


Figure 114 – Surviving generations and garbage collection of InternetGasturbine web application

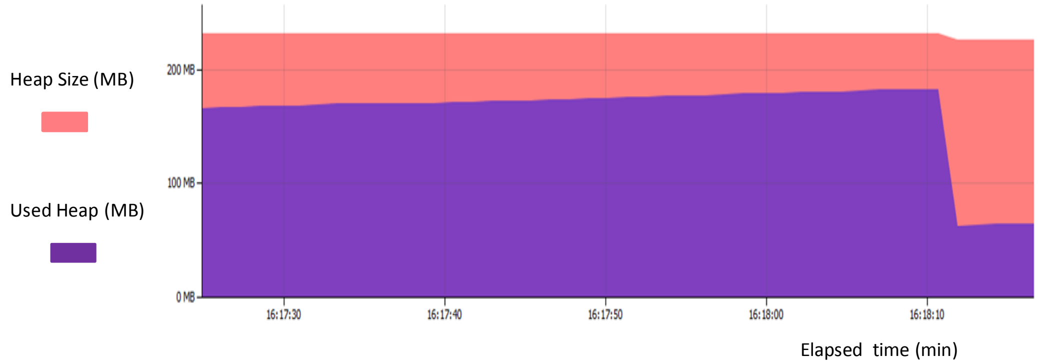


Figure 115 – Heap size and usage of InternetGasturbine web application

The web application and the revised Java SE modules were also analysed for quality metrics, in order to investigate clarity, maintainability and reusability. The separation in individual specialised software components has prevented excessive number of Lines of Code (LOC) in one program. The core modules, EISI and MEOCS had an abstractness of around 0.54, indicating good extensibility and reusability. An average instability (I) of 0.17 was also observed in EISI and MEOCS, indication of encapsulation, reusability and maintainability. It can also be seen that Lack of Cohesion of Methods (LOCM) was maintain relatively low, in an attempt to achieve cohesion in the designed classes. Avoidance of complexity was also achieved, indicating understanding of the applied algorithm. Evidence of low complexity was the low average values of Cyclomatic complexity (VG), Nested Block Depth (NBD) and Weighted Methods per Class (WMC). Abstractness was not important for the web application, as the REST web services were designed for retrieval of particular resources. The two last programs in Table 18 were simple single class applications, hence the relatively higher NBD can be explained, as they contained all there logic in the main method.

Table 18 – Quality metrics of designed software components

	LOC	NCP	A	I	LOCM	NBD	VG	WMC
EISI	1385	6	0.54	0.17	0.34	0	1	109
MEOCS	1263	4	0.5	0.17	0.33	0	1	54
WebApp	475	6	0	1	1.11	0	1	23
RestartServer	74	1	0	0	1	2	1	4
Monitor App	104	1	0	0	1	2	1	5

The provisions that were implemented with regards to the Internet connection disruptions have addressed one of the main elements of the risk analysis for round 4. The components dependencies were also considered and with the existence of alternative approaches, the required functionality was eventually obtained.

9 ACCEPTANCE VALIDATION

This chapter describes and discusses the final round (round 5) of the Spiral software process, which includes the general high level validation of the integrated Internet application. It adheres to the principles of the Spiral approach, with definition of the objectives and the risk assessment of the process prior to any operation.

9.1 Objectives – Risk Assessment

The objectives that guided this round are shown in Table 19

Table 19 – Objectives of round 5

Revision and finalisation of the System Requirements
Identification of the various states of the application
Derivation of test cases according to the application states transition
Conduction of high level Requirements Validation based on formal methods
Ensure total Statement and Decision coverage testing of the application
Conduction of a preliminary probabilistic analysis of reliability of the introduced software components

Risk assessment of this stage was also accomplished with HAZOP, with procedural risks only considered. This round was not an implementation round as the previous ones hence there were newly introduced risk elements, regarding the validation methodology and approach to this process. The most noticeable risks were related to the validation method selection, the derivation of the proper test cases and the coverage of the validation. These were all issues that could occur due to the restriction of time, lack of experience and ambiguities of the system requirements. The suggestions for mitigation included revisiting and finalising of the requirements - according to the functionality that was eventually accomplished and to conclude which features should be suspended for later development. Additionally, research was recommended for formal validation approaches in other safety critical software systems and if possible, in the aviation industry as well.

The risk assessment finally addressed the clarity of the objectives once again, along with the cost, which was also considered in all the previous rounds. In this round, the cost could have increased if a non open source software tool was selected for the validation - as a result of lack of experience - or the application of a method unsuitable for the current project (Appendix B, Section B.7).

9.2 Methodology

This section was divided in two sub–sections. The first describes the methodology applied for the requirements and final testing procedures of the acceptance validation, whilst the second focuses on the methodology applied for the probabilistic analysis of reliability.

9.2.1 Requirements testing and final testing methodology

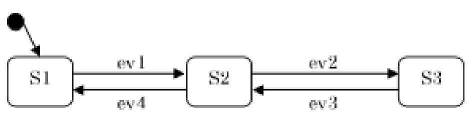
The distinction between requirements testing and final testing was based on the fact that the former covered the satisfaction of the system requirements whilst the latter tested functionalities not predicted by the system requirements, but were necessary to be tested in order to obtain 100% statement and decision coverage of the application. The term 'final' testing is also used by Hinderman et. al. (2007) [69]. They were both accomplished in parallel, integrated under the term "Acceptance Validation".

Model checking was selected for the conduction of the acceptance validation. As discussed in 3.5.5, this is a formal verification technique [11] and it is commonly applied in critical software systems. In contrast to simulation, formal verification methods do not rely upon the dynamic response of the system, but perform static analysis on a formal mathematical representation of the system, in order to check it for correctness with respect to a given specification. Model checking has been gaining popularity in aerospace systems design and verifications the last years [4]. Gargantini et.al. [50] - in their work for a specification-based method for constructing a suite of test sequences, where a test sequence is a sequence of inputs and outputs for testing a software implementation – suggest that model checking may be used for the derivation of test cases for black – box specification testing of an application. It is a powerful approach for the formal verification and validation of software. It automatically provides complete proofs of correctness, or explains, via counter-examples, why a system is not correct [11]. Hence it was selected due to the ability to depict the application with a robust mathematical representation and derive precise test cases, instead of relying on the accuracy of simulation approaches.

NuSMV 2.5.0 was selected as the model checking tool for the general validation. NuSMV is an open source symbolic model checker branched from SMV (Symbolic Model Verifier). The advantage of SMV against explicit model checking according to Kim et. al (1999), is that the latter allows free variables to represent environmental inputs, instead of modelling the system environment explicitly, as required in the former [83].

NuSMV has been developed jointly by FBK-IRST and by Carnegie Mellon University [37]. It allows for representation of synchronous and asynchronous finite state systems, and for the analysis of specifications expressed in Computation Tree Logic (CTL) and Linear Temporal Logic (LTL) [25]. From the discussion in 3.5.5, it was concluded that CTL can be highly exhaustive in path analysis, which was a priority for this project hence it was manly applied, apart from very few cases, where the investigation of a possibility transition to particular states along time was questioned.

The generation of test cases was accomplished with an approach widely met in automated test case generation applications with NuSMV. An example of this approach can be seen in the work presented by Kadono et. al., about using the NuSMV Model Checker for test generation from statecharts [80] (Figure 116). In their approach the test cases were generated from the counterexamples produced by NuSMV, with regards to the failed specifications. They applied CTL with the use of three temporal operators: AG, AX and EF: AG holds in state s if g holds in all states along all computation paths starting from s. AX holds in state s if g holds in all next states to s and EF holds in state s if g holds in some state along some computation path starting from s [80].




```
MODULE main
VAR
  pc : {S1, S2, S3};
  ev : {ev1, ev2, ev3, ev4};

ASSIGN
  init(pc) := S1;
  next(pc) := case
    pc = S1 & ev = ev1 : S2;
    pc = S2 & ev = ev2 : S3;
    pc = S2 & ev = ev4 : S1;
    pc = S3 & ev = ev3 : S2;
    1 : pc;
  esac;
```

Figure 116 – Statechart and NuSMV program [80]

For this project, the integrated system was assumed to be a finite UML state machine with the complex internal processes as black boxes, which were consequently modelled in individual finite state machines. All the derived state machines were used to define the corresponding NuSMV variables and the specifications to be tested. The negation of the tested specifications has driven the model checker to produce analytic counterexamples, from which the test cases were derived. In this project, AG, AX and EX were applied for CTL. According to the NuSMV manual [25], EX holds in a state s if there exists a state s′ such that a transition goes from s to s′ and the condition is true in s′. As for LTL specifications, G, F and X formulae were applied, which, according to the NUSMV manual, are defined as shown below:

- G: p is true at time t if p is true at all times t′≥t
- F: p is true at time t if p is true at some time t′≥t

- X: p is true at time t if p is true at t+1

9.2.2 Probabilistic analysis methodology

The methodology applied for the probabilistic analysis of reliability relied on Markov equations, which is an approach based on a state diagram and applicable in risk assessment of power systems [87]. Li (2005) describes it as extremely useful for modelling the outage of individual components, but difficult to be applied in large systems. Indicatively, a system of N components of 2 states each would require 2^N states to be modelled. A finite Markov chain may be defined as a finite Markov Process for which the transition probabilities p_{ij} *(n)* do not depend on *n,* hence denoted p_{ij}. A Markov chain is described by the transition Matrix P with entries p_{ij} [82]. A Markov chain with a discrete set of times is known as discrete time Markov chain (DTMC) [36]. Gokhale et. al. (2002) [54] have indicated a classification of DTMC's in two categories:

- Irreducible: A DTMC is said to be irreducible if every state can be reached from every other state.
- Absorbing: A DTMC is said to be absorbing, if there is at least one state *i,* from which there is no outgoing transition. A DTMC upon reaching an absorbing state is destined to remain there forever.

Trivedi (1982) [139] describes how the transition matrix P of an absorbing DTMC for a system with *n* states can be reduced to a new *(n-1) x (n-1)* sub-stochastic matrix Q with at least 1 row sum less than 1, describing the probabilities of transition only among transient states, and a column vector C ().

$$\boldsymbol{P} = \left[- - \frac{\boldsymbol{Q}}{\boldsymbol{0}} - - \middle| - - \frac{\boldsymbol{C}}{\boldsymbol{1}} — \right]$$

Equation 10 – Reduced transition matrix

Consequently, the k step transition probability matrix can be written as:

$$\boldsymbol{P^k} = \left[- - \frac{\boldsymbol{Q^k}}{\boldsymbol{0}} - - \middle| - - \frac{\boldsymbol{C'}}{\boldsymbol{1}} — \right]$$

Equation 11 – k step transition probability matrix

where C' is a column vector with elements of no further use. The *(i,j)th* entry of matrix Q^k denotes the probability of arriving in transient state s_j after exactly *k* steps starting from also transient state s_i. As described by Trivedi (1982) [139], it can be shown that $\sum_{k=0}^{t} Q^k$ converges as t tends to infinity, implying that the inverse matrix $(I\text{-}Q)^{-1}$ exists. This is called the fundamental matrix M and it is a source of very useful information about the DTMC. Let V_j be the average number of executions of a state s_j in a run of the program. Since execution commences at state s_1 the average times of execution (or times visited) for the j_{th} state would be:

$$V_j = m_{1j}$$

Equation 12 - Average number of executions of a state sj

The general approach to the reliability analysis of this application has followed the concept of the work presented by Gokhale et. al. (2002) [54], which referred to reliability prediction and sensitivity analysis, based on software architecture. The study of Gokhale et. al. (2002) [54] included 3 sections. The first has used the mean time of execution and the mean number of visits of each state to determine performance of the system, whilst the second used the same parameters to define the reliability. The third section of the work included a sensitivity analysis of performance and reliability against variation of time spent in each component, and the alteration of mean visits, which essentially depicts the workload of the components. Having assumed reliabilities of individual components to be known, Gokhale et. al. (2002) [54] stated that the overall reliability of the examined system for single execution is given by:

$$\boldsymbol{R} = \prod_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{R_i^{X_{1,i}}}$$

Equation 13 - Overall reliability for single execution

The number of visits to each component is a random variable, implying that reliability also is a random variable. With the use of Taylor series they concluded that the expected reliability of the system is given by:

$$\boldsymbol{E[R]} = \left[\prod_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \left(\boldsymbol{R_i^{m_{1,i}}} + \frac{1}{2}\left(\boldsymbol{R_i^{m_{1,i}}}\right)(\boldsymbol{log R_i})^2 \boldsymbol{\sigma_{1,i}^2}\right)\right] \boldsymbol{R_n}$$

Equation 14 - Expected reliability of a system

By ignoring the second order architectural effects Equation 14 can be reduced to the following expression:

$$\boldsymbol{E[R]} \approx \prod_{\boldsymbol{i=1}}^{\boldsymbol{n-1}} \boldsymbol{R_i^{m1,i}} \boldsymbol{R_n}$$

Equation 15 – Reduced form of expected reliability of a system

From the results of the study of Gokhale et. al. (2002) [54], it was observed that the difference of reliability estimation was of 3rd decimal magnitude. Thus for the preliminary reliability analysis of the designed application, Equation 15 was considered to be adequate and hence applied. When the failure rate is constant, the reliability function can be expressed exponentially in terms of time *t* and failure rate ϑ [29]:

$$R(t) = e^{-\theta t}$$

Equation 16 – Reliability function

The above approach for reliability estimation was also applied from Meedinya et. al. [93], in their study about architecture-based reliability evaluation, under uncertainty of software systems. In their study they have included the consideration of workload, the computational load of software components executing a requested task, the execution initiation probability, the data size and the next step probability. They also considered two sources of failure, the execution failures and the communication failures. One of the additions to the Gokhale et. al. (2002) [54] approach was the implementation of Monte Carlo simulation to draw samples from the probability distributions of the input parameters, in order to achieve the required quantitative metric of the quality of the system. A dynamic stopping criterion terminates the Monte Carlo simulations when sufficient samples have been taken to achieve the desired accuracy. At a later stage, Meedeniya et al have extended their work by application not only to reliability evaluations but also to a variety of probabilistic properties [94].

Another concept that was considered for the reliability analysis of this application was the work of Littlewood (1997) [89], who has introduced conservative stopping rules of operational testing for critical safety systems. Littlewood has examined demand based systems and continuous time systems. For the current application, the demand based systems and particularly the proposed pfd-based stopping rule. This approach refers to the probability of failure on demand (pfd) for systems or applications of a system that is only required to function under a certain set of inputs. For the currently tested application, the function of shutting down and refreshing of the servers upon network connection loss, was examined with the pfd stop rule proposed by Littlewood. Littlewood (1997) [89] has assumed that successive demands are statistically independent Bernoulli trials. Let *p* be the probability of failure on demand. Thus, given *p*, the number of failures in *n* demands, reliability *R* has a Binomial distribution:

$$\boldsymbol{P(R = r) = nCr\, pr\, (1 - p)n - r}$$

Equation 17 – Reliability in terms of pfd

And particularly:

$$P(R = 0) = (1 - p)_n$$

Equation 18 – Probability function of R=0 in terms of pfd

Within the Bayesian framework one may represent a priori knowledge about the parameter of interest, here p, by the prior distribution:

$$f(p) = \frac{p^{a-1}(1-p)^{b-1}}{B(a,b)}$$

Equation 19 – Beta distribution function of parameter p

Where *B(a,b)* is the Beta function and *a>0, b>0* are chosen by the observer of the experiment to represent his belief about p prior to seeing any test results. If the system has executed n demands, and has seen r failures, the posterior distribution of p is *Beta(a+r,b+n-r),* with a=b=1 due to the lack of previous existing information:

$$f(p|r, n, 1, 1) = \frac{p^{r}(1-p)^{n-r}}{B(1+r,1+n-r)}$$

Equation 20 - Beta distribution function of parameter p with a=b=1

Littlewood (1997) [89] then computed *n1* by asking what was the minimum number of demands that, if executed without failure, would allow to conclude that the system had met its pfd target. For *P≥0.99*, equation 16, in a more generic form becomes:

$$P(\mathbf{p} < \mathbf{p_0}) \geq \mathbf{1} - \boldsymbol{\alpha}$$

Equation 21 - Probability of failure p to be less than a required value p_0

From Equation 20, n1 is the smallest value of n for which the following equation is satisfied:

$$\int_0^{p_0} \frac{(1-p)^n dp}{B(1,1+n)} \geq \mathbf{1} - \boldsymbol{a}$$

Equation 22 - Condition for having the smallest value of runs n_1

If the system is placed on test and failure actually occurs after s_1 *(<n_1)* demands, then n_2 is computed, the number of further demands that must now be executed failure-free to satisfy the reliability requirement, as follows. The posterior distribution for *p* immediately following the failure on the s_{1th} demand is:

$$f\,(p|1, s_1, 1, 1) = \frac{p(1-p)^{s1-1}}{B(2,s_1)}$$

Equation 23 - Beta distribution function of parameter p after failure at *s1* demands

The above has now become the prior distribution for *p* for the further testing to be conducted. To compute n_2, the posterior distribution after seeing n_2 further demands, all executed failure-free, is required. This is:

$$f\,(p|1, s_1 + n_2, 1, 1) = \frac{p(1-p)^{s1+n2-1}}{B(2,s_1+n2)}$$

Equation 24 – Calculation of new value of runs n_2

This process continues. In general, if the *jth* failure has just been seen, and the failures occurred on the s_{1th}, $(s_1+s_2)_{th}$, . . . , $(s_1+s_2 \ . \ . \ . +s_j)_{th}$ demands, a further n_{j+1} demands should be required to be executed failure-free, where n_{j+1} is the smallest value of *n* for which:

$$\int_0^{p_0} \frac{p^j (1-p)^{\sum_1^j s_i + n - j} dp}{B\left(j+1, \sum_1^j s_i + n - j\right)} \geq \mathbf{1} - \boldsymbol{a}$$

Equation 25 - Condition for having the new smallest value of runs nj+1

9.3 Requirements testing and final testing

The main state machine sm0 of the application was derived from the mode 0 of the functional requirements (4.2.3), describing the basic functionality of the InternetGasturbine web application. The various states and transitions were defined with regards to the functional requirements included in this mode. The nested individual requirements represent a state transition. The main state machine was then used as a dashboard from which several less abstract state machines were derived to represent the other 5 modes of the functional requirements. Consequently, the state machines became the framework for 6 corresponding NuSMV models. The NuSMV specifications were built in a manner to describe sequential transitions between states and their negation produced the required test cases for each individual functional requirement. There were a total of 168 test cases derived. In several occasions there were test cases applicable in more than one requirement. The details of the analysis for each mode individually are shown in Appendix D.

For mode 0 the start state and the end state were identical (servers running and ready to accept) (Appendix D, Section D.1.1). The transitional state *run* was branched to state machine sub1 (Appendix D, Section D.1.2), which describes the states and transitions of *mode 1and 2*: the engine start functional requirements. This state machine was further more analysed to sm2 (Appendix D, Section D.1.4), describing mode 3 (engine running), sm3 (Appendix D, Section D.1.4) describing mode 5 (engine stop), sm4 (Appendix D, Section D.1.5) describing mode 4 (trouble shooting) and finally sm5 (Appendix D, Section D.1.6), which represents specifically functional requirement 4.3 of mode 4 (Figure 117).

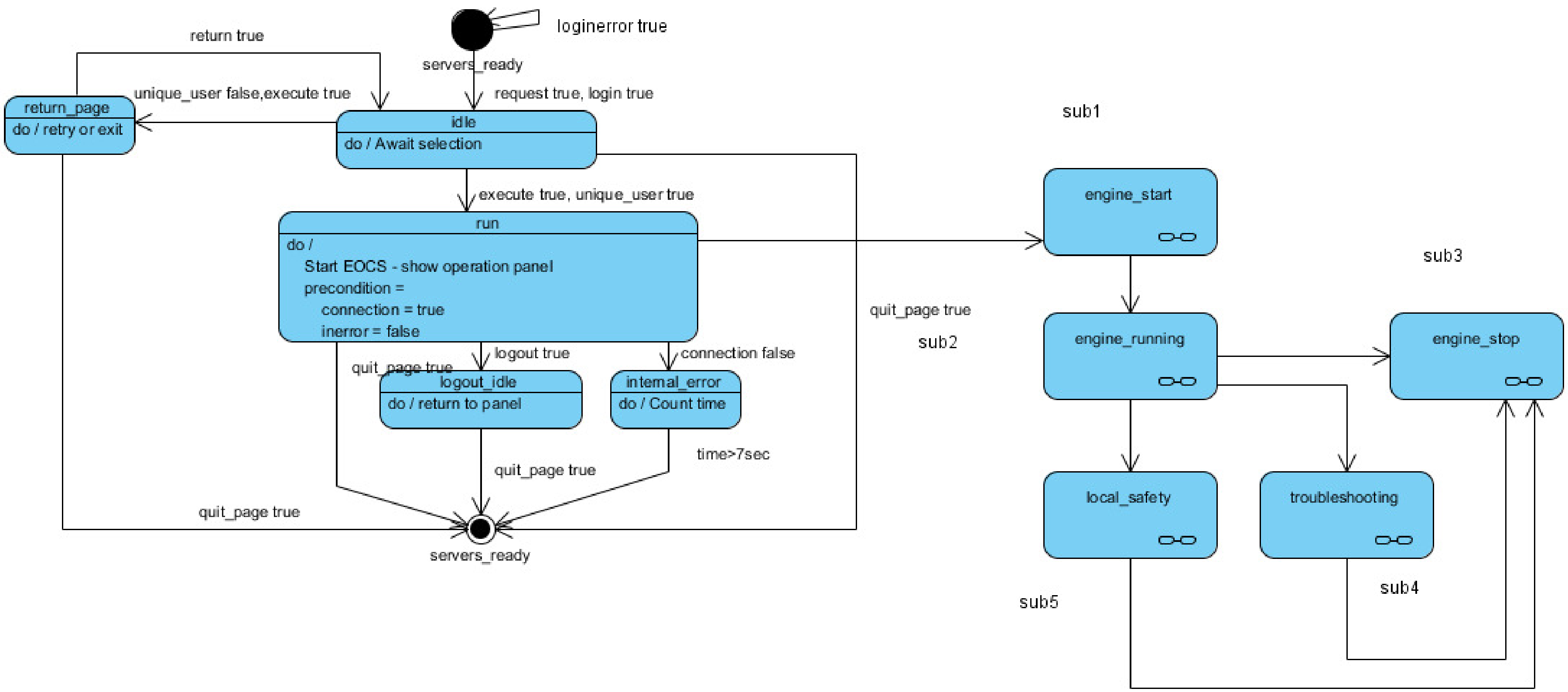


Figure 117 – State machine for mode 0 of System Requirements

The designed NuSMV models indicated thoroughly all the potential state transitions of the integrated application. In order to increase the coverage and in pursuit of 100% statement and decision coverage, there were extra requirements added in the validation tables, beyond the functional requirements of the SRS document. These extra requirements were denoted as 'additional' in the tables. The testing of the extra requirements was accomplished with the derived test cases as well.

9.4 Probabilistic analysis of reliability

The application was investigated for the expected reliability of the data acquisition and command transmission functionalities. The data acquisition is triggered by requests from the client to the methods that connect to the assigned ports. The reception of a data package is acknowledged by the reception method on the MEOCS side and the response to the transmitting socket (EISI side) initiates the transmission of the next package. The response ends as reception confirmation to the ECU or the NI modules connected with the other sensors. The hardware components and the ECU were assumed as one single integrated component for this analysis, as their reliability is mostly known from the manufacturers and has been addressed in risk assessment during the previous rounds. The command request was also assumeed to end up in the same integrated hardware component. The system was assumed as a absorbing finite DTMC.

In order to apply Equation 15, the individual reliability of the each components was required. There was lack of sufficient information of prior distribution due to the fact that the components were newly designed. Hence the approach to establish individual reliabilities was empirical with a total time of operation around 100 hrs, although not continuous. The observed failures were expressed as a failure rate of failures per hour. The failure rate was assumed to be constant hence Equation 16 could be applied to estimate reliability of individual components. For the components that showed no failures, they were assumed to have a failure rate of 10^{-4}, as generally agreed for a practical limit of a failure rate in the range of 10^{-2} /h or 10^{-4}/h [60], discussed in 3.4.3. Faults that were repeated during start up of the system (mainly due to LabVIEW delay on the very first run each time the system started) were not considered, as they would not reappear if the system was running continuously. The components which the derived DTMC comprised are shown in Table 20 and Figure 118.

Table 20 –The components of the derived DTMC

Component	Description	Reliability
s1	Reverse Proxy	0.99
s2	Glassfish Web Server	0.99
s3	Command REST service	0.99
s4	Engine data REST service	0.99
s5	Peripheral data REST service	0.99
s6	MEOCS command thread	0.99
s7	MEOCS engine data thread	0.99
s8	MEOCS peripheral data thread	0.99
s9	EISI command thread	0.99
s10	EISI engine data thread	0.99
s11	EISI peripheral data thread	0.99
s12	EIS switch signal generation	0.99
s13	EIS throttle signal generation	0.99
s14	EIS data reception	0.37
s15	ACAS data reception	0.37
s16	Integrated hardware	0.99

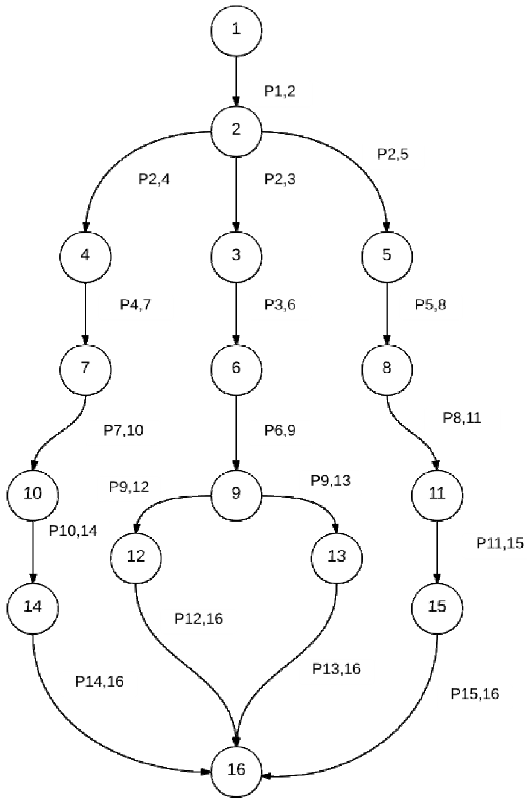


Figure 118 – DTMC of the application

The process was considered from the Reverse Proxy onwards (Figure 118), as the client computer was a remote independent component, not directly linked with the internal reliability of the application. When a request reaches the Glassfish server, the request will be dealt with through 3 possible routes, as they are directed through the same port, hence processed one at a time. The command request could be distributed even more, as it could be either sent to the switch or the throttle generation LabVIEW function. The probabilities of transition are shown in Table 21 – Transition probabilities between the components of the DTMC.

Table 21 – Transition probabilities between the components of the DTMC

$p_{1,2} = 1$		
$p_{2,3} = 0.33$	$p_{2,4} = 0.33$	$p_{2,5} = 0.33$
$p_{3,6} = 1$		
$p_{4,7} = 1$		
$p_{5,8} = 1$		
$p_{6,9} = 1$		
$p_{7,10} = 1$		
$p_{8,11} = 1$		
$p_{9,12} = 0.5$	$p_{9,13} = 0.5$	
$p_{10,14} = 1$		
$p_{11,15} = 1$		
$p_{12,16} = 1$		
$p_{13,16} = 1$		
$p_{14,16} = 1$		
$p_{15,16} = 1$		

The transition probabilities were assumed to be equally distributed in every distribution node. This would depend on the actual workload of the components but as there was no experimental data yet available, equal probabilities were assumed. The model was then tested for sensitivity to the variation of the transition probabilities, after the distribution nodes. From Table 22, a 16x16 table transition matrix *P* was derived, from which the reduced 15x15 *Q* matrix was defined. Consequently, the inverse matrix $(I\text{-}Q)^{-1}$ was calculated, providing the fundamental matrix *M.* According to Equation 12, the elements of the first row of *M* represent the average times of visit of the respective components (Table 22).

Table 22 – Reliability and average number of visits of components

Component	Average number of visits (m_i)	Individual Reliability (R_i)	Expected Reliability ($R_i^{m_i}$)
s1	1	0.99	0.999
s2	1	0.99	0.999
s3	0.2	0.99	0.998
s4	0.4	0.99	0.996
s5	0.4	0.99	0.996
s6	0.2	0.99	0.998
s7	0.4	0.99	0.996
s8	0.4	0.99	0.996
s9	0.2	0.99	0.998
s10	0.4	0.99	0.996
s11	0.4	0.99	0.996
s12	0.18	0.99	0.998
s13	0.02	0.99	0.999
s14	0.4	0.37	0.672
s15	0.4	0.37	0.672
s16	1	0.99	0.999

From Equation 15 (having ignored the second order architectural effects), the expected reliability of the system was calculated (Table 22).

A crucial feature of the designed application was the capability to detect a delay from the client to request data, and consequently shut down the engine and the program safely. This feature was considered to be tested with the proposal of Littlewood (1997) [89], described in 9.2.2. With the assumptions adopted in Littlewood's approach, within the Bayesian framework, assumed desired pfd $<10^{-3}$, and $P = 1\text{-}\alpha \geq 0.99$, the implementation of Equation 22 has produced a minimum value of $n1$ = 4602.

9.5 Results – discussion

The System Requirements had been through the last revision during round 4. After that stage the feasibility of all the requirements was reconsidered and tasks that were time demanding or involved major redesigns were suspended for future development. One of the most noticeable functions not accomplished was the design of a database installed on the server connected to the gas turbine. The database was planned to include files with performance parameter recordings, faults, errors and operation time. As this task would extend the completion time of the project with a negative knock on effect on the thorough validation process, it was postponed to be implemented at a later stage. Revisit of the System Requirements is one of the main characteristics of the Spiral Model that distinguish it from other more concrete models, such as the Waterfall model. After the initial draft of the System Requirements was released, every round included the inspection of feasibility through the objectives determination and appropriate revision of the requirements until their final form after round 4.

The application, as a total, had a high level of complexity, which was a restrictive factor in representation of the system with a single state machine. Hence the derivation of simpler sub state machines was inevitable. The functional requirements were used as a guideline for the sub state machine classification. The functions described by each functional requirement were presented as nested state machines within the main state machine (sm0), describing the state transitions of within the InternetGasturbine web application. This approach analysed the system in sub-systems of significantly less complexity, with their internal states easily distinguished and defined.

The technique of negation of the NuSMV specifications produced explicit counterexamples (Figure 119) that outlined the required procedures for inter-state transitions. The counterexamples were then matched to the corresponding nested functions within the examined functional requirements, to provide the required test cases and validation tables (Appendix D, Sections D.2, D.3). The detailed representation of the state transitions allowed for the observation of several functionalities that were not included in the functional requirements hence provided additional test cases to capture all the possible paths of execution of the application. Prior to the acceptance test, during every individual round of the Software Process, the unit testing and the integration testing had ensured 100% statement and decision coverage of the individual modules. Thus the acceptance validation essentially was an approach to integrate and revise all the previous testing. At this point, the desired 100% statement and decision coverage of the total application was achieved.

```
-- specification AG (state = idle -> !(EX state = ready))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 5.1 <-
  state = ready
-> Input: 5.2 <-
  loginerror = FALSE
  unique_user = FALSE
  login = TRUE
  logout = FALSE
  request = TRUE
  execute = FALSE
  quit_page = FALSE
  interror = FALSE
  connection = FALSE
-> State: 5.2 <-
  state = idle
-> Input: 5.3 <-
  loginerror = FALSE
  unique_user = TRUE
  login = TRUE
  logout = FALSE
  request = FALSE
  execute = TRUE
  quit_page = TRUE
  interror = FALSE
  connection = FALSE
-> State: 5.3 <-
  state = ready
```

Figure 119 – NuSMV counterexample

The reliability analysis was conducted in a very short period of operating time (100 hrs). The gas turbine and interface hardware components were assumed as an integrated component with known reliability. This assumption was necessary in order to consider the application as a whole. Further analysis of the components would result in a very complicated DTMC model, not assessable. One of the known disadvantages of Markov analysis is that it cannot be applied to complicated systems hence the target of this analysis was to demonstrate the simplest possible approach to study the reliability of the designed application through a DTMC model.

The short period of running the application revealed occasions of Input / Output exceptions in the ServerEIS thread and the AmbientIn thread at the EISI side, during the start up of the application. The problem was raised by the delay (magnitude of around 1000 of ms) of the LabVIEW EIS and ACAS programs to instantiate. The issue was alleviated with the insertion of a 500 ms delay in the instantiation sequence prior to the EIS start up. These problems were encountered once during the runs hence the individual reliabilities of the EISI engine data and ACAS data threads were estimated from Equation 16 to be 0.37. These 2 components were the bottlenecks of the application due their low reliability. Consequently the total system Expected Reliability, calculated from Equation 15 was found to be:

E[R] = 0.483

The DTMC model underwent sensitivity analysis to the variation of the workload (expressed in terms of transition probability) after component 2 (Figure 118). Initially the transition probability $p_{2,3}$ was varied and, $p_{2,4}$ and $p_{2,5}$ were assumed equal. In the second case, $s_{2,4}$ was considered to be the control variable and the other two assumed equal. The analysis (Figure 120) has shown that increasing the probability of transition to s_3 would increase the expected Reliability, as that branch includes components with relatively high reliabilities. Increased transition to s_4 or s_5 (bottlenecks) would reduce the expected reliability. However, the absolute gradient of Reliability variation after linear regression indicated a higher sensitivity (improvement) when transition is favoured towards the flawless branch through s_3, rather than when the branches with reduced reliability were preferred. This fact can be attributed to the independence of the paths (independent threads) that does not allow multiple interactions between the individual components, thus increasing the overall expected Reliability of the system.

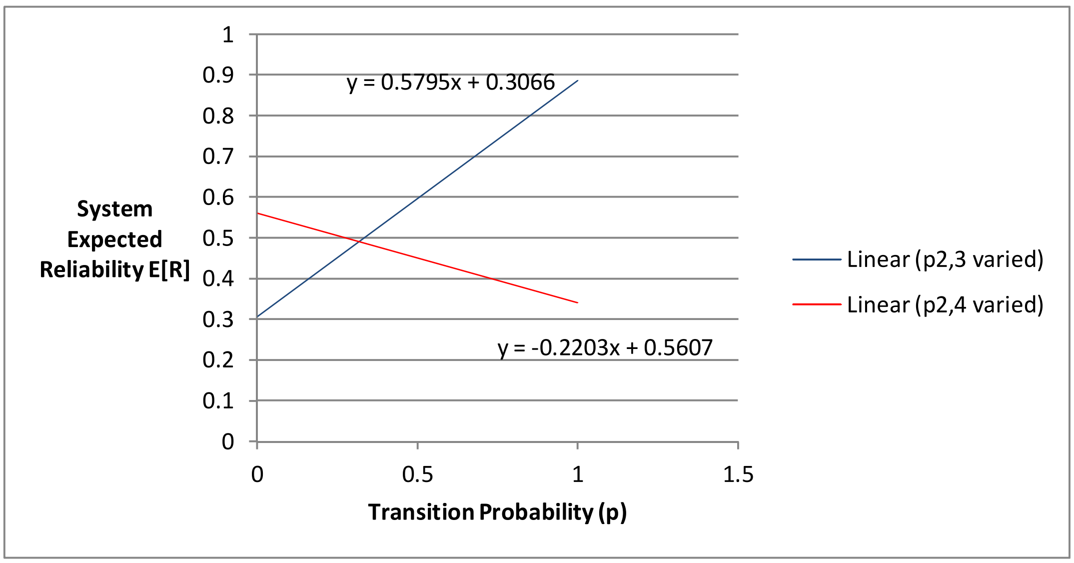


Figure 120 – Sensitivity analysis with probability transitions s2, 3 and s2, 4 varied

The sensitivity of the model to the variation of the individual reliability of one single component was also examined. One of the less reliable components was considered as the control variable. The improvement of the total expected Reliability was significant, with a gradient of around 0.52 (Figure 121), after linear regression.

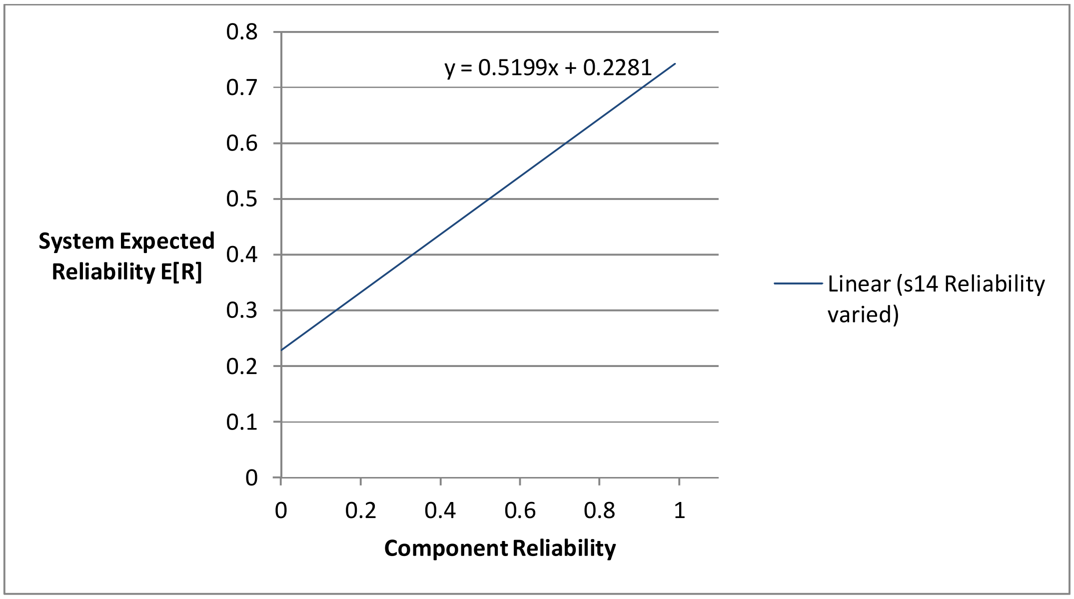


Figure 121 - Sensitivity analysis with Reliability of s14 varied

Finally, the minimum required runs (calculated from Equation 22) to test the reliability on demand of the network connection loss functionality were not able to be accomplished during the development of the Thesis (over 4,500 runs required). At the time of writing, this function required further testing with the release of a beta version of the application.

In order to obtain a more detailed Reliability estimation, the components would require a separation into nested sub components, in order to obtain simple DTMC's for smaller groups. This approach would be applicable after a sufficient amount of time of operation, when necessary data will have been collected. Additionally, the software modules may be detached from the hardware components in order to allow a separate reliability evaluation for the reusable software and for the tightly coupled hardware equipment. The total Reliability of the integrated system may depend on the hardware reliability as well but in a generic approach with different gas turbines, the reusable software components of the application should be investigated for expected Reliability independently from the hardware components.

Finally, all the risks of this round identified in the risk assessment (Appendix B, Section B.7) were successfully encountered, and the round was concluded with a successful acceptance validation process where the desired coverage was achieved and no extra costs addition to the project.

10 DEVELOPMENT CAPABILITIES

This chapter presents the capabilities of the application for future development and interaction with other applications. The suggested modifications are presented in an abstract manner and the final section of the chapter discusses some potential applications on the long term. The subject of this chapter has resulted from the non functional requirement of generic capabilities of the application, set in the SRS document. There was no concrete planning for this requirement hence all the possible generic configurations were discussed herein.

10.1 Addition of different gas turbines

In order to attach a different gas turbine, the main modification required would be the adaptation of the LabVIEW EIS (Engine Interface Software) module. The TSS (Troubleshooting System) would only require a different input file to contain the nominal values and standard deviations of the critical parameters for the new gas turbine. Additionally, the user interfaces for both LAN and web versions would need to be modified according to the gas turbine features. In Figure 122, the required modifications are shown, classified by the cluster of components where they belong. Each software module was depicted as a UML component and then the packages and classes (or LabVIEW functions) to be modified are shown. In some cases the hierarchy was presented down to the methods of the classes to be modified. For the web application, the web pages were shown within the highest level folder that contained them. The modifications are explained in Table 23.

It can be noticed from the modifications layout that the LabVIEW EIS component was the only module tightly coupled with the gas turbine at the back end. The majority of the components could be reusable with any other gas turbine type. The components that require modifications were very few, proportional to the total application, with limited changes.

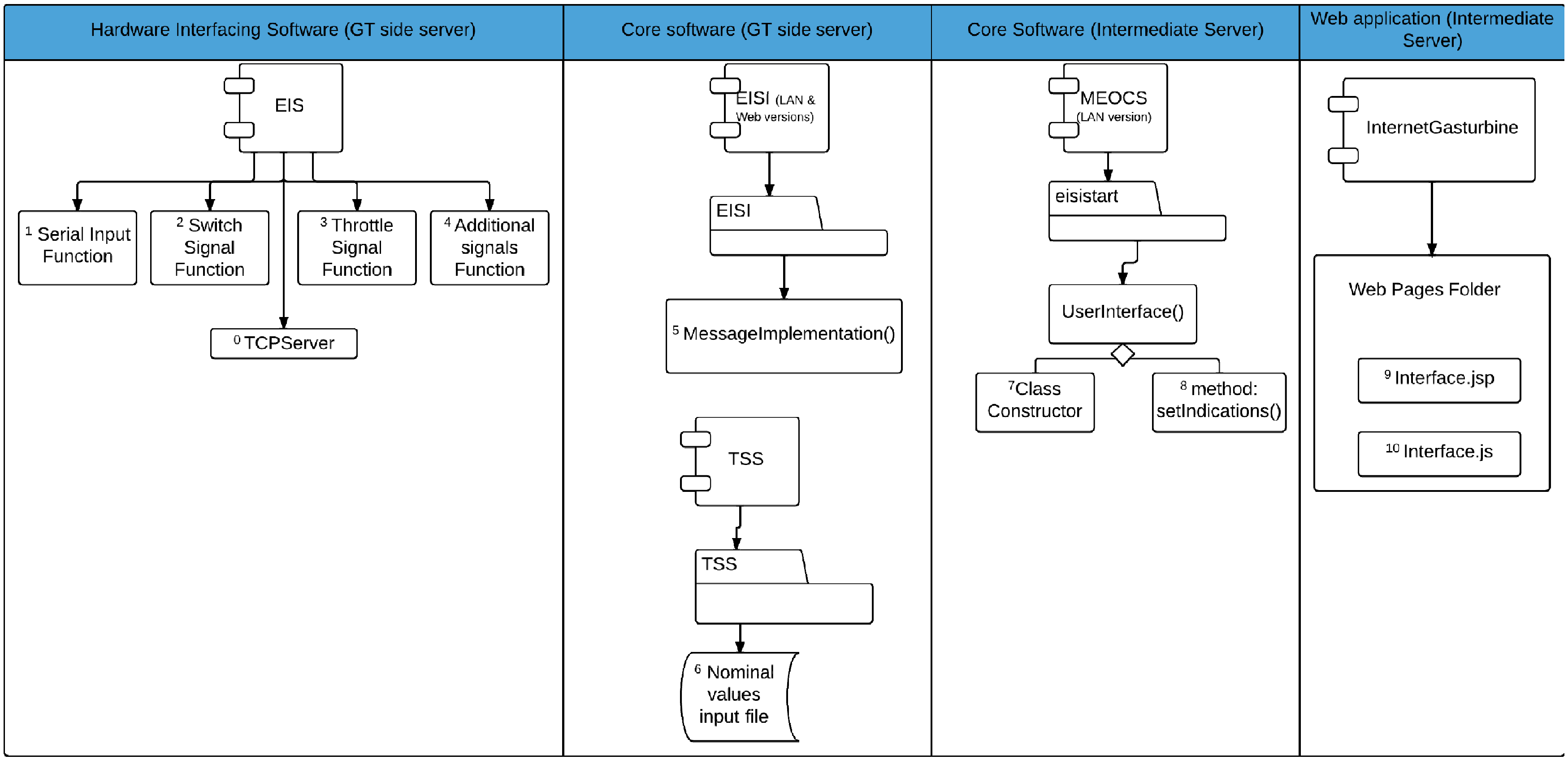


Figure 122 – Modifications required to accommodate a different gas turbine

Table 23 – Modifications legend for accommodation of a different gas turbine

Index	Modification Description
0	Array of input storage increased number of indexes as required – Nested case structure to include additional cases
1	Function altered according to the engine control unit and sensors output. Could be other than serial input: analogue or digital.
2	Function altered to produce required signal according to the new gas turbine.
3	Function altered to produce required signal according to the new gas turbine.
4	Additional functions may be added if more than one signals are required
5	All method definitions maintained as indicated by MessageConfiguration() interface. Re-implemented according to engine output format.
6	New file added with values corresponding to new gas turbine
7	Modified to include necessary controls and indicators
8	Modified to update new indicators
9	Modified to include necessary controls and indicators
10	Modified reading() function to update new indicators

10.2 Configuration with multiple test stations and gas turbines

The application was designed with the ability to accommodate several gas turbines simultaneously. The commands are sent to the engine sequentially and the system currently can accept 100 commands defined only by an index number (flag) assigned to them by the user interface. The interpretation and allocation takes place at the last software component before the engine ECU, which, as discussed in 10.1, is the only component that is tightly coupled with the gas turbine (the EIS module). Additionally, the application can accommodate a gas turbine installed in an integrated test cell, provided that the data produced by the sensors in the engine and the peripheral hardware is distributed in the 2 major groups of data: one for the gas turbine readings (regardless of their number) and the other for the data acquired from the ambient and peripheral sensors.

There were 2 approaches considered for a multi-engine configuration. Option A suggested the design of individual web applications for each connected gas turbine.

Each web application would import an instance of the MEOCS, which would then invoke the appropriate server at the gas turbine end (Figure 123). This option would provide a more flexible design, presenting a more REST-compatible API and reduced demand per unit time for each web application (in the case of multiple gas turbines operating simultaneously). Thus, this option would favour increased reliability on demand.

Option B proposed a single web application, which should include libraries with all the necessary operation panel pages and the corresponding JavaScript files (Figure 124). The MEOCS here would be used as a dashboard which connects the client to the desired gas turbine at the back end. Although this approach may be more prone to fail in case of excessive simultaneous demand, it considered to be adequate, as the application is not expected to be accessed simultaneously by more than a few tens of users. The main advantages of this option are that it would be more efficiently secured under a single entity (one web application) and also that the access to all the individual engines could be monitored and controlled centrally, with one or two Servlets.

The modifications required (Figure 125) are expected to be identical for both options, apart from the web application, which in Option A it would contain only those pages required for the single related gas turbine, whereas in Option B the web application would contain a library of gas turbine pages. From the modifications list (Table 24), it can be seen that the system may easily be configured, without major redesign.

It is noticeable that the AMT Netherlands Olympus used in this project is configured to accept electric signals from the ECU to manipulate the controls. If a differently configured gas turbine was connected, there would be an additional requirement to install actuators to the controls, in order to convert the electric signals to mechanical transition. Although in contemporary gas turbines the control unit may override the mechanical signal to control the throttle position, there are certain features that cannot be overridden (for extra safety purposes), such as in the PW-F100 engine, where a mechanical switch pressed by the throttle lever during movement enables fuel flow above the idle position.

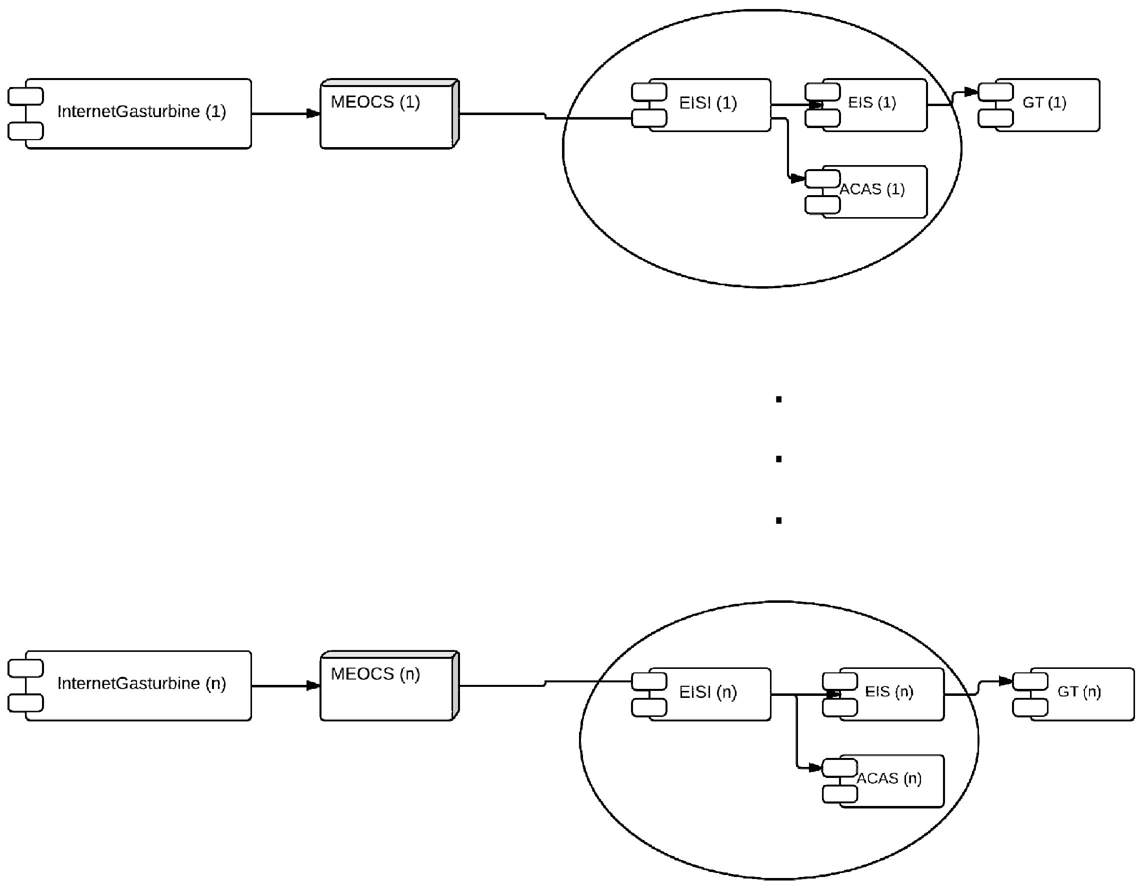


Figure 123 – Multi GT configuration – Option A

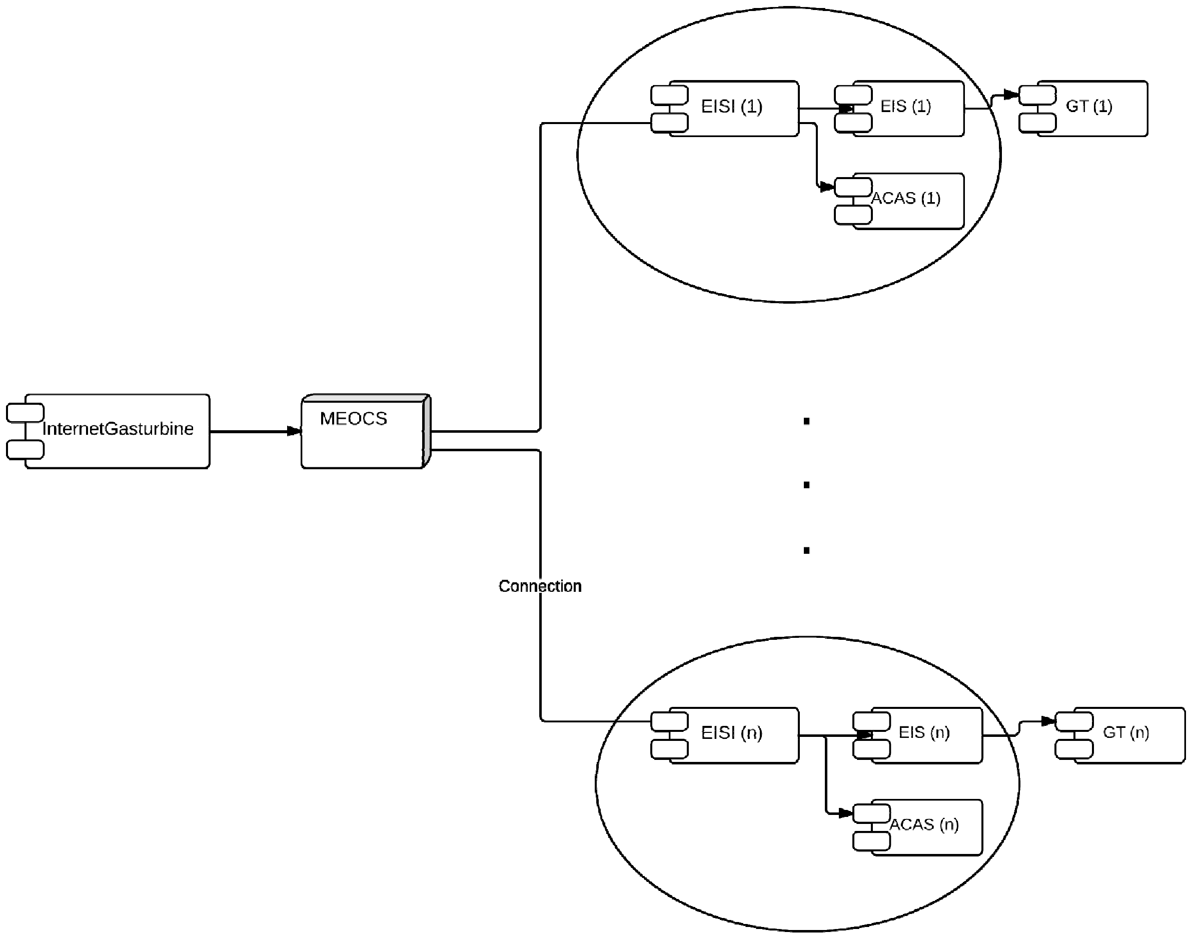


Figure 124 - Multi GT configuration – Option B

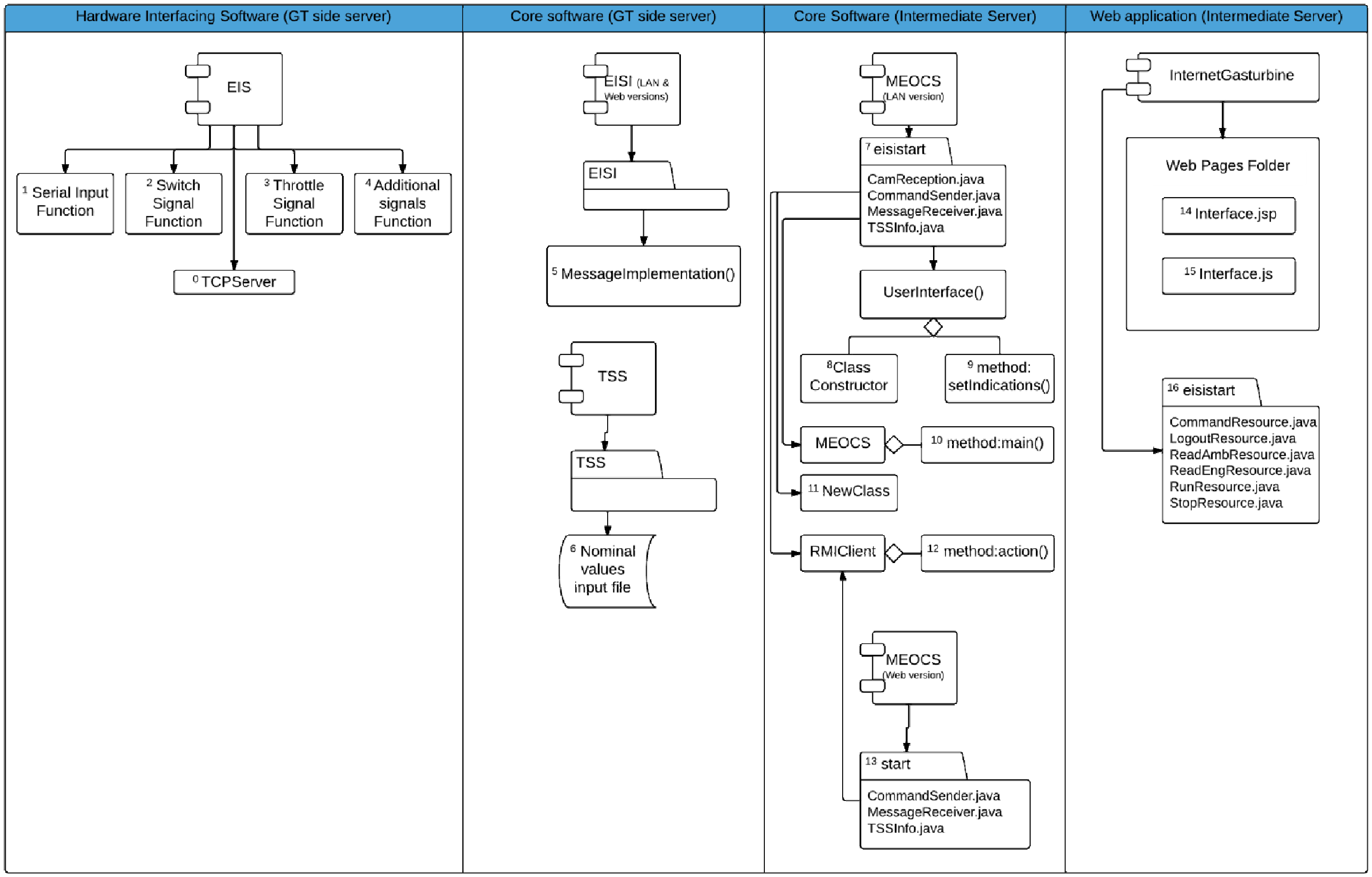


Figure 125 - Modifications required to accommodate multiple gas turbines

Table 24 - Modifications legend for multiple gas turbines configuration

Index	Modification Description
0	Array of input storage increased number of indexes as required – Nested case structure to include additional cases
1	Function altered according to the engine control unit and sensors output. Could be other than serial input: analogue or digital.
2	Function altered to produce required signal according to the new gas turbine.
3	Function altered to produce required signal according to the new gas turbine.
4	Additional functions may be added if more than one signals are required
5	All method definitions maintained as indicated by MessageConfiguration() interface. Re-implemented according to engine output format.
6	New file added with values corresponding to new gas turbine
7	Add gas turbine type index in the constructors of the listed classes – Add list with sockets and ports
8	Modified to include necessary controls and indicators - prototype class cloned accordingly
9	Modified to update new indicators - prototype class cloned accordingly
10	Current main method declassified as main – identified as static
11	New class introduced to contain main method and landing GUI. Selection of gas turbine as an event to invoke the static method of class MEOCS with the corresponding index
12	List with addresses of remote methods, sockets and ports included
13	Add gas turbine type index in the constructors of the listed classes – Add list with sockets and ports
14	Library of jsp operation panels (for Option B only)
15	Library of JavaScript functionality pages (for Option B only)
16	All root methods of listed RESTful web services to accept gas turbine index parameter from the operation panel

10.3 Integration with the WebEngine

Interaction of the Internet Gas Turbine with the TURBOMATCH engine would provide an integrated experimental platform that would allow the user to adapt his created performance simulation model to the actual gas turbine, based on real time engine data. TURBOMATCH is a very powerful performance simulation tool and the WebEngine enables the creation of models remotely, through a regular web browser. The validation of a model would always be easier and more accurate if the user could make direct comparisons of simulated and real output accomplished against the real engine. A simulation model many times is based on ideal conditions and assumptions thus it cannot simulate accurately every single gas turbine of the same type. Therefore, adaptation of a created model to actual readings from a particular gas turbine would enhance the accuracy of the model.

The proposed collaboration of the two applications would be feasible with simple additions. A RESTful root resource needs to be added to the InternetGasturbine application, such that would it accept requests from the WebEngine client. Consequently, the root resource would invoke a local data acquisition method, which would utilise the already existing methods to retrieve the data from the engine and the environment. With PCN as the handle of the model, TURBOMATCH would require the PCN of the gas turbine and any other available measurement – in this case the EGT, thus T_6. PCN could be calculated from the InternetGasturbine if the design point rotational speed is known Equation 26

$$PCN = RPM/RPM_{DP}$$

Equation 26 – Gas turbine shaft speed

Multiple samplings would allow the acquisition of valid values, clean from any instantaneous peaks or drops. Hence after a loop of sampling, the average values could be returned to the REST resource and consequently back to the WebEngine client (Figure 126).

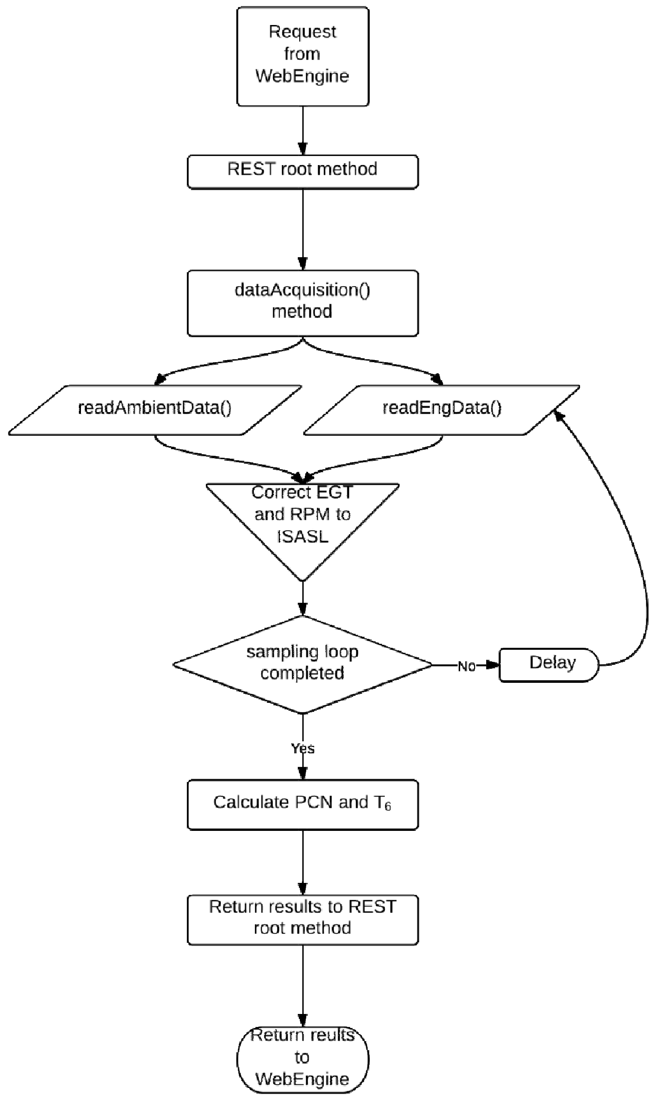


Figure 126 – Retrieval of data from the Internet Gas Turbine

A different approach was also suggested, with which the accuracy of the TURBOMATCH model could be examined at Off Design conditions (Figure 127). This time, the settings of the model should be passed to the actual engine, which consequently will adjust the RPM according to the setting of the PCN handle of the model. A restriction posed here would be the ambient conditions where the engine is operating, hence the model must be designed in conditions to match those of the engine environment – T_{amb} and P_{amb} set as the values acquired from the engine environment. When the InternetGasturbine receives the request, the method that conveys the commands will be called iteratively until the actual engine PCN (Equation 26) has approached the model PCN sufficiently. The criterion for this would be the value of the absolute difference Δ_{PCN} between the obtained engine PCN and the model PCN. Convergence would occur if Δ_{PCN} achieved a value less than a very small number ε, of magnitude 10^{-3} (Equation 27).

$$\Delta_{\mathrm{PCN}} = |PCN_{Obtained} - PCN_{Model}| < \varepsilon$$

Equation 27 – Convergence of gas turbine PCN to TURBOMATCH model value

After convergence has been achieved, the InternetGasturbine would return the performance match matrix V, a column vector which in this study would comprise 2 elements, PCN and T_6 (Equation 28).

$$V = \begin{pmatrix} \frac{PCN_{obtained} - PCN_{model}}{PCN_{model}} \\ \frac{T_{6\,obtained} - T_{6\,model}}{T_{6\,model}} \end{pmatrix}$$

Equation 28 – Performance match matrix

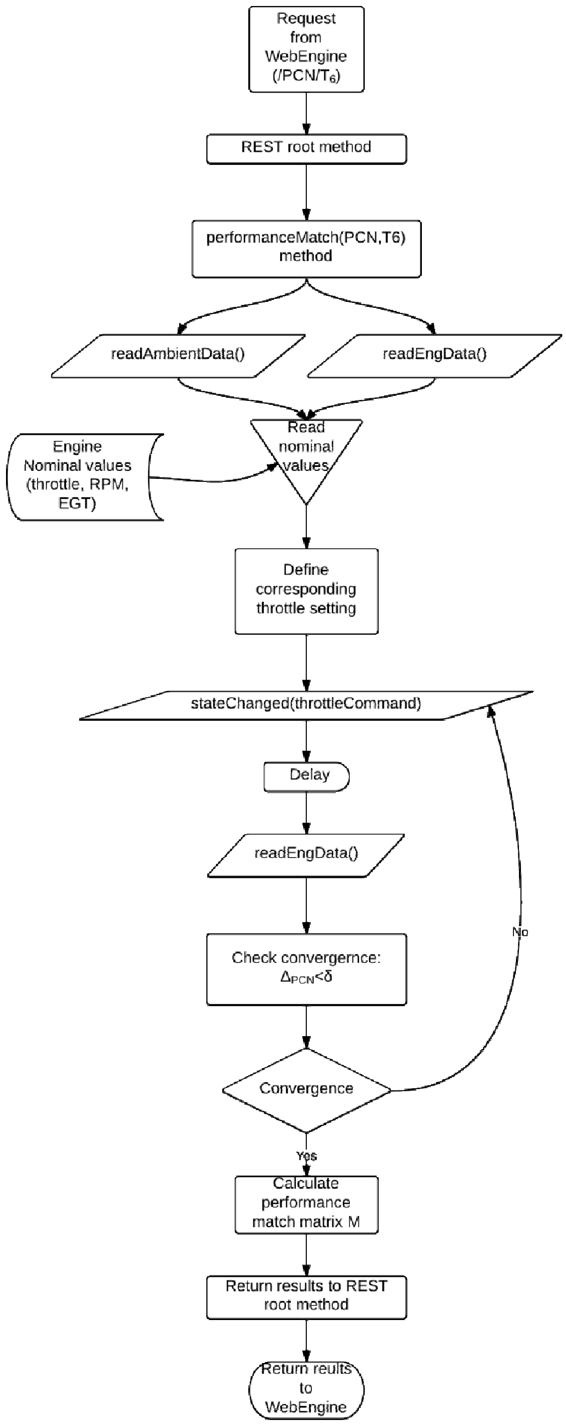


Figure 127 – Adaptation of engine state according to the WebEngine model

The first approach was simulated with an experiment that involved the design of a simple Java EE 6 web application, where a web page including an HTML form, sent AJAX POST requests after a button event to the InternetGasturbine and acquired the engine and ambient data. The data were displayed in dedicated fields of the web page. A restriction that was encountered was the fact that AJAX does not allow cross domain requests. The problem was overcome by installation of the 2 applications in the same domain, as virtual hosts under the same virtual server of the configured reverse proxy. Although the main applications HTTP containers (Glassfish) were installed in different computers, the AJAX requests were accomplished flawlessly. Practically, instead of assigning a clean URL to each of the applications (www.webengine.com and www.Internetgasturbine.com), they should be accessed under a central common host:

www.common-host.com/webengine

and

www.common-host.com/Internetgasturbine

Another technique for enabling cross-domain requests is JSON-P. In 2005 JSONP (later dubbed JSON-P, or JSON-with-padding) was formally proposed as a way to leverage this property of <script> tags to be able to request data in the JSON format across domains. It works by making a <script> element (either in HTML or inserted into the DOM via JavaScript) that requests to a remote data service location. The response is the name of a function pre-defined on the requesting web page, with the parameter being passed to it being the JSON data being requested. When the script executes, the function is called and passed the JSON data, allowing the requesting page to receive and process the data [51]. However, JSON-P has essentially been a loose definition by convention, when in reality the browser accepts any arbitrary JavaScript in the response. This means that authors who rely on JSON-P for cross-domain Ajax are vulnerable potential threats, as a malicious web service could return a function call for the JSON-P portion, but slip in another set of JavaScript logic that could hack the page into sending back private user's data [51]

The capability of configuration of the InternetGasturbine to be accessed by multiple observers – WebEngine clients simultaneously and acquire or transmit data, was considered. This situation was successfully simulated by creating a new jsp page containing the gas turbine operation panel, with the controls omitted. The IneternetGasturbine application could only start one instance of the server side operation control modules (as expected) but the observer clients could monitor the operation without interventions. The observers’ browser clients could invoke the MEOCS static methods for data acquisitions and the server was not affected by any closure actions or network disruption from the observers. Only the central user who had instantiated the process could shut down the application.

The main drawback observed, was the inability of the application to detect a disconnection of the central user, as the server side data acquisition methods were constantly invoked by the observers and were not allowed to time out. To overcome this weakness, it is suggested that a ping function was added to the main operation panel web page. This function would transmit ping requests to the server in predefined time intervals (5 sec would be sufficient), hence if the main user has lost connection, the server would be informed and proceed to the shutdown of the application, regardless if the observers are still connected.

10.4 Practical Applications

The integration of the Internet Gas Turbine application and the TURBOMATCH WebEngine would form a powerful experimental tool to be used from overseas students and engineers. It would also provide a useful tool for preliminary estimation of performance of modelled gas turbines with similar architecture to those included in the application. Additionally, the combined application could be used as a central performance and health monitoring station, from where the user can run gas turbines of his fleet remotely and establish a performance base line to monitor each gas turbine individually. The application would also enable the monitoring and testing of aircraft installed engines remotely, from specialised staff at the base, without the need to transfer them at the location of the aircraft and the engine.

In a more generic version, the Internet gas turbine application could be developed to operate UAV's or drones through the Internet. There are 2 approaches for this option. The easiest approach would be to control a conventional remote control console through the Internet. The final software module in this case would produce and interpret appropriate radio signals rather than generating electrical signals, and reading physical signals from the ECU (Figure 128). However, the most challenging approach would be to utilise the capability of Wi-Fi connection in flight, which is a currently applicable and constantly expanding feature [53]. In this case, the back end server and the engine side modules (EISI and EIS) would be installed on an appropriate device with Wi-Fi connection on board the UAV. These would then be connected to the necessary hardware modules, interfacing the control unit of the vehicle (Figure 129).

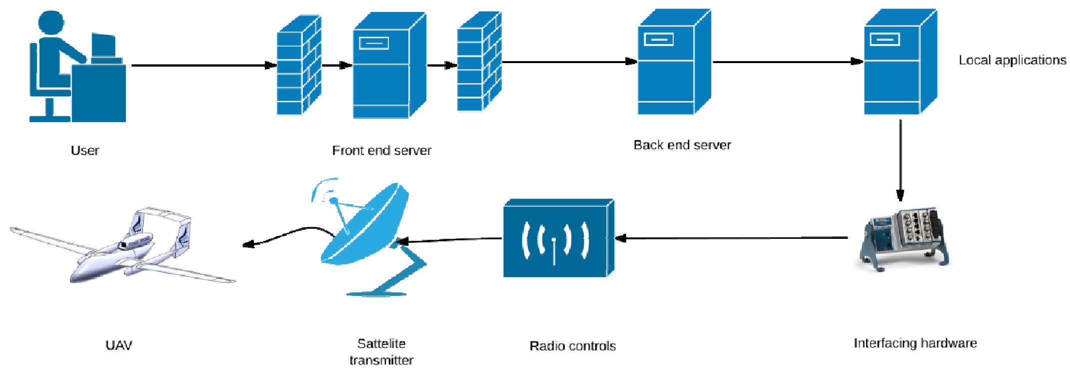


Figure 128 – UAV operation through conventional radio controls

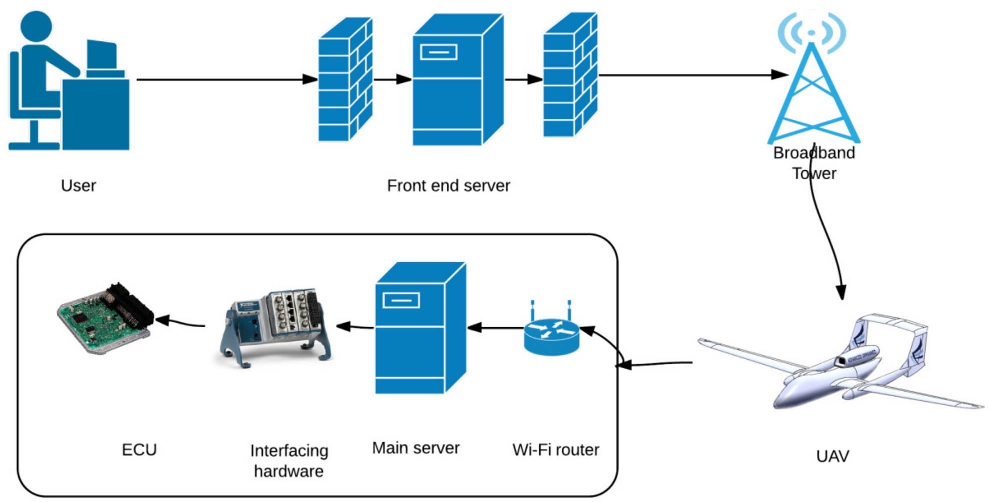


Figure 129 - UAV operation through on flight Wi-Fi connection

11 RESULTS - DISCUSSION

This chapter summarises and discusses the obtained theoretical and experimental results of the project. Experimental values were not analytically presented herein, as they have been presented and discussed in the Results section of chapters 4-9. The optimisation of different aspect of the application was also considered here.

11.1 SRS – General risk assessment

The System Requirements Specification (SRS) was constructed in accordance with standard IEEE-1012-2004. Following the elicitation of the functional requirements, the general risk assessment of the project was conducted. The most common risk identified was the loss of power supply at the test house where the gas turbine is installed, accompanied by a combination of other undesirable individual events that included several types of Internet-based threats. Two main end events were identified and all the individual event combinations were isolated with the application of minimal cut sets. Eight minimal cut sets were produced for the first end event and 16 for the second. The ETA that followed indicated that the worst potential consequence would be major damage of the gas turbine, without involvement of personnel injuries; hence the risk of the project was assessed as acceptable. Subsequently, a prototype program was developed and used to indicate the feasibility of the proposed application. The acceptable general risk and the feasibility of the functional requirements concluded the feasibility study of the system. The aforementioned tasks were a part of round 0 of the Spiral Model software process. Moreover, the designed prototype program was utilized in round 2 for the construction of the General Test Harness of the application.

11.2 Implementation

The implementation phase was composed of rounds 1 to 4 of the Spiral approach. During these rounds, the gas turbine was connected and interfaced with a computer, the core module of the operation control software was designed and, the LAN and web prototypes were completed.

11.2.1 Round 1

The local interfacing produced an adjacent PC operation station with a LabVIEW interface, with all the available parameters of the engine successfully acquired. There was 100% match of the RPM and also less than 1% deviation in EGT acquisition; the deviation was caused due to the different accuracy of the engine EDT and the LAbVIEW indications. A slight deviation between the generated throttle signal and the throttle slider position was observed, which was attributed to the fact that the required signal was analogue, hence not feasible to obtain precise correspondence between slider and signal. The deviation was restored at throttle settings above 10% and had no effects to the operation of the system. The function that read in the engine data produced a cyclomatic complexity VG = 20, much higher to the conventionally acceptable value of 10. This value was attributed to the high number of output messages from the engine and the necessity to translate them into plausible indications, resulting in the design of complex case structures in LabVIEW. However, the function was thoroughly tested with unit testing techniques and treated as a black box from then onwards.

11.2.2 Round 2

Round 2 of the Spiral focused on the design of the core program that would control the operation, by interacting with the previously designed LabVIEW Engine Interface Software (EIS). The cyclomatic complexity of the created modules was maintained at the value of 10 and lower. Individual methods were tested with unit testing techniques and the overall integrated system up to that point was successfully tested at 100% statement and decision coverage. The examination for potential memory leaks caused by the Java SE modules revealed satisfactory functionality of the automatic garbage collection procedure, with maximum momentary time spent in garbage collection found around 0.1%. The trend of surviving generations of objects was maintained constant after initial build up and peaks. When the two components (MEOCS and EISI) of the core software module were installed in separate computers within the LAN of the University campus, a latency of around 200ms for the command to arrive at the EISI was observed. The latency was negligible when the components were installed together. This was the inevitable effect of the physical impedance and the latencies in the network stack, which could not be avoided.

11.2.3 Round 3

In round 3, the Troubleshooting System (TSS) was designed and implemented, enabling real time monitoring of the gas turbine operation and providing automatic handling of any parameter exceedance. This feature has downgraded the importance of latency between the modules and also mitigated one of the most noticeable observed underlying risks: the mains supply power loss. The fuel flow signal was captured but as no reference values were available it is not acquired with great accuracy. Accuracy could be improved by calibration of the transducer after completion of this project.

The cyclomatic complexity of the modules was maintained below 10 and the surviving generations of objects of all the Java components followed a stable trend after initial peaks. Under no circumstances the time spent in garbage collections exceeded 0.1%. The quality metrics of the Java modules presented satisfactory results. The separation in individual specialised software components has prevented excessive number of

Lines of Code (LOC) in a single program. The core modules had an abstractness of 0.54, indicating good extensibility and reusability. An average instability (I) around 0.25 was also observed in the EISI and the MEOCS, and 0.0 in the TSS, implying encapsulation, reusability and maintainability. It can also be seen that Lack of Cohesion of Methods (LOCM) was maintained relatively low, in an attempt to achieve cohesion in the designed classes. Avoidance of complexity was also achieved, indicating understanding of the applied algorithm. Evidence of low complexity was the low average values of cyclomatic complexity (VG), Nested Block Depth (NBD) and Weighted Methods per Class (WMC).

By the end of this round a fully operational LAN version of the applications was completed.

11.2.4 Round 4

The web application was successfully tested in 5 major browsers and 4 operating systems, with 4 different versions of Windows. The JavaScript functionality has performed as expected in all the testing environments. The application was configured to run over SSL with cache disabled. It was secured with a form based authentication and configured with a reverse proxy at the front end. To enhance security, it was proposed for the reverse proxy to be accessed through the University VPN.

The embedded flash player required the shockwave flash player plugin to be activated in the browser. In order to increase the independence of the application from third party plugins or add-ons, it would be recommended to embed the HTML5 version of flowplayer as well in the operation panel web page and allow the user to make a dynamic selection of which form to enable. The compromise to this would be the fact that HTML5 would not be compatible with older versions of Internet Explorer (<8).

There was a high percentage of unit testing of the implemented methods and the control flow testing of the various modules achieved 100% statement and decision coverage during validation. The cyclomatic complexity of the JavaScript modules was maintained in low levels. Also, the Java EE web application was found to perform with a low number of surviving generations with trend that stabilised within the first minutes of operation. The code quality metrics of the Java SE components have not been affected by the inevitable adaptations. Low abstractness observed in the Internet Gas Turbine web application was not important, as the RESTful web services were designed as individual resources, which could be called from other applications in future development.

11.3 Validation

The functional requirements structure provided the framework for the requirements testing. Model checking provided the necessary test cases to test each individual function of the functional requirements. For this reason the operation of the application was represented with state machines. This was feasible with the consideration of less complicated sub-machines to express each functional requirement individually. The state transitions within the derived state machines revealed functionality that was not predicted the SRS document. The additional functionality was also tested in order to obtain maximum coverage of validation. A total of 110 test cases were derived and all the possible state transitions were tested (Figure 130). Every module and integrated group of components had already been tested for 100% statement and decision coverage, hence with the requirements - final testing the complete application could be considered to have achieved that goal as well. The validation in stages was one of the advantages of an iterative software process, such as the Spiral Model, which was applied for this project. It would not be feasible with a linear model software process to achieve such a high coverage of validation with the current application, due to the very high cyclomatic complexity if it was considered as an integrated application, testable only upon overall completion.

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
0.1.1	Sm.0	0.2	TH - AP	OK	
0.1.2	Sm.0	0.1	TH - AP	OK	
0.1.3	Sm.0	0.3	TH - AP	OK	
0.1.4	Sm.0	0.3	TH - AP	OK	
0.1.5	Sm.0	0.8 – 0.9	TH - AP	OK	
0.1.6	Sm.0	0.6 – 0.7	TH – AP	OK	
0.1.7	Sm.0	0.6 – 0.8	TH – AP	OK	
0.1.8	Sm.0	0.4	TH – AP	OK	
0.1- additional-1	Sm.0	0.5	TH – AP	OK	
0.1-additional-2	Sm.0	0.10	TH – AP	OK	
0.1-additional-3	Sm.0	0.11	TH – AP	OK	
0.1-additional-2	Sm.0	0.12	TH – AP	OK	
0.1-additional-3	Sm.0	0.13	TH – AP	OK	
0.2.1	Sm.0	0.3	TH – AP	Flowplayer error 201: Stream not found	Renew Wowza Media Server 3.6.2 License
1.1.1	Sm.1	1.16	TH – AP	OK	
1.1.2	Sm.1	1.16	TH – AP	OK	
1.1.3	Sm.1	1.16	TH – AP	OK	
1.1.4	Sm.1	1.16	TH – AP	OK	
1.1.5	Sm.1	1.16	TH – AP	OK	
1.1.6	Sm.1	1.16	TH – AP	OK	
1.1.7	Sm.1	1.16	TH – AP	OK	
1.1.8	Sm.1	1.16	TH – AP	OK	
1.1.9	Sm.1	1.16	TH – AP	OK	
1.1.10	-	-	TH – AP	-	Suspended
1.1.11	Sm.1	1.16	TH – AP	OK	

Figure 130 – Excerpt from the acceptance validation tables

The reliability analysis was based on experimental data obtained from a very short period of operation during the acceptance validation phase. Although the results could not be considered totally representative of the Expected Reliability (discussed in Section 9.4), the approach was an analytical demonstration of reliability assessment with the application of DTMC models. With the proposed assumptions, the Expected Reliability was found to be:

E[R] = 0.483

The value of the Expected Reliability seems low but it is not representative, as sample runs were very few (100) hence 1-2 faults that occurred affected the overall reliability noticeably. However, this was an indicative value that established a point of reference to conduct a sensitivity analysis that demonstrated how the alteration of the workload of the individual components could affect the total reliability. As the main paths of the operating function were designed to be independent, only transition towards those that included bottlenecks, would affect the reliability negatively.

The minimum required runs to test the reliability on demand of the response to network connection disruption, were calculated at a number around 4,600. This result occurred by adapting the example of Littlewood (1997) [89] for pfd<10^{-2} applied in a nuclear station control software, which is considered as a safety critical system application. The runs could not be accomplished during the development of the Thesis and required further testing after a beta version release of the application.

11.4 Installation and data handling

The LabVIEW components were built with version 2011 SP1 for Windows 7 64 bit operating system. It is noticeable that all the latest versions of LabVIEW do not receive full support for all toolkits and drivers in operating systems other than Windows (Windows 7 and Windows Server). The Java modules were built with JDK 1.7.17 for Windows 64 bit, however the packages could easily be cloned in different JDK versions for other operating systems, such as 32 bit Windows or Linux. Similarly, the Java EE 6 modules could be adapted for other operating systems. Hence the main restriction in operating platforms was posed by the LabVIEW components, which cannot run on other OS except Windows 7 with full capabilities. The above restriction implied that the EIS, the EISI and the TSS should be installed on Windows 7 64 bit, as they are all required to be on the same computer. The MEOCS and the InternetGasturbine web application could be configured for different OS, provided that they were both designed for the same platform.

The application can be easily distributed in 2 compressed folders (.zip or .rar): One must include the executable files for the 2 LabVIEW programs – EIS and SignalClosure (ACAS is invoked within the EIS.exe file) - and the jar files for EISI, TSS, Restartserver and WatchDog Java programs. The compressed files must be extracted in the same folder. The second compressed folder must include the InternetGasturbine web application .WAR file, which should be extracted and deployed in a Glassfish server instance. The .WAR file has already imported an instance of MEOCS during build. As for the Apache Virtual host, this must be added in the httpd-ssl configuration file of an Apache server. Analytically, the size of the required files for the LAN and the web versions are listed in Table 25.

Table 25 – Size of execution files

Module (executable and configuration files)	**Size**		
	Local	**LAN**	**Web**
EIS	488 KB	706 KB	706 KB
SignalClosure	N/A	460 KB	460 KB
WatchDog	N/A	16 KB	16KB
RestartServer	N/A	12 KB	12 KB
EISI	N/A	2.76 MB	717 KB
MEOCS	N/A	2.72 MB	307 KB
InternetGasturbine	N/A	N/A	3.56 MB

It can be observed from Table 25 that the size of the required files was not significant and if the components were installed separately, then the space occupied on the disk would be even more reduced.

The transfer of real time data was handled with Buffered Readers set at their default size (8,192 characters in Java) and TCP as the transport layer protocol. The data was handled satisfactorily with not any necessity to increase the size of the buffers. The exchanged data was of 0.1 KB magnitude and not more than 200 characters. It can be seen that with the default size of the buffers much larger packets of data could be successfully handled with very low latency values. The request – response latency of course depends on the network quality and the distance as well. Operation within the campus network showed an average latency of 14.29ms of the requests to be executed and return a response (Figure 131), with all the components and servers installed in the same computer. Analysis of an average latency value request (Figure 132) has shown that the amount of time waiting was around 14ms and the execution 2ms. The observation of latency variation would become more detailed after release and trial of the application under actual circumstances, something that was not possible during the development. However, as latency is inevitable, the implemented real time monitoring of the engine operation (TSS) reduces the significance of the delay in the data transfer.

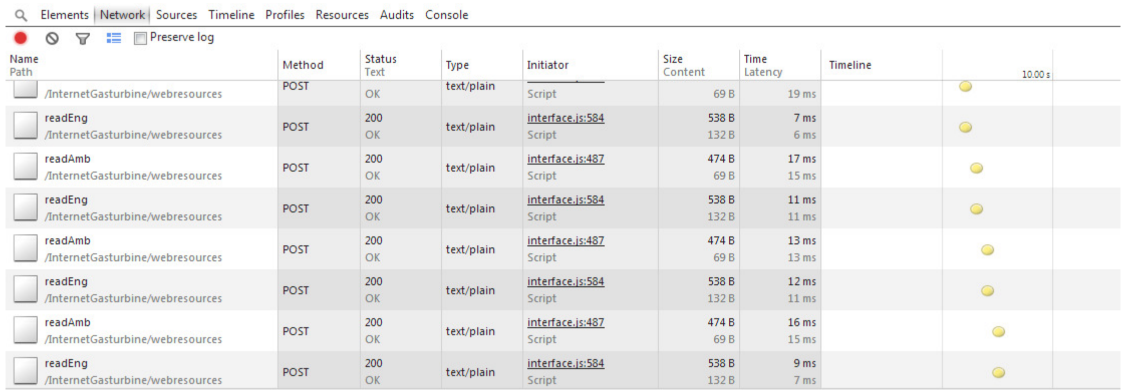


Figure 131 – Request latency

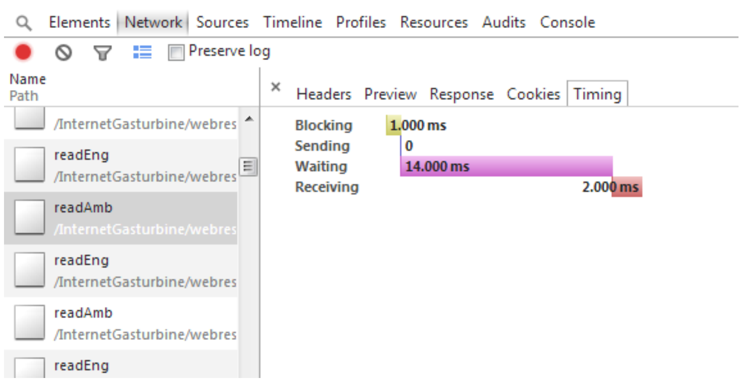


Figure 132 – Latency analysis

Details of the installation procedure, prerequisites and execution of the application are outlined in Appendix F.

11.5 Optimisation of different aspects of the application

The application was optimised mainly for safety and security. It was designed for 2 additional human observers to monitor and interfere if necessary, although their presence was not requisite for the operation of the system. The placement of the local PC observer required the extension of the LabVIEW engine interface (EIS) in order to accommodate total functionality for controlling the operation if necessary. The extension of the EIS has increased the load of the CPU and added latency to the data flow. On the contrary, it has enhanced safety, as Internet operation may be overridden at any time by the observer. The operation of the gas turbine was not very flexible, as the system does not accept rapid throttle increase (>20%). Higher settings increase could only be achieved incrementally with inputs less than 20%.

Access to the system is limited and authorisation to register can only be granted by the administrator locally at the server end. This implies a restricted number of users and an application not accessible unless permission granted by an authorised administrator. Also, the configuration for operation over SSL has raised some issues with online access to remote libraries, such as YUI. A small delay was inevitably added to the loading time of the web page.

Reliability was another element highly considered, hence the implementation of TCP protocol for real data transmission. TCP caters for the quality of the transmitted data packages without the necessity for overhead design on behalf of the developer. Additionally, the data packages were checked at every reception point and if not found in the appropriate form, the application reused the last valid package received.

To optimise for performance, the following modifications would be required:

- Abolishment of the observer station. This would reduce the workload of the local computer, as engine output would not require to be processed for local display. Consequently, the response time to the remote user would be reduced. To compensate for the reduction of observers, it would be recommended to design and install an automated transition from remote to manual operation mode. This would require a signal generation to be provided to an automatic relay switch to be activated to change the source of the signals towards the ECU (from computer to manual control unit). Moreover, the default position of the relay switch when not loaded could be configured to switch to the manual control unit, upon power supply loss.
- Implementation of UDP for the transport layer where real time data is involved. This would reduce the latency of data exchange between the different layers. The disadvantage for this approach would be the necessity for the verification of the quality and the order of the received data, with possible consequences in reliability.
- Installation of all the components in the same computer. This approach would reduce any latency added due to the internal network quality and impedance. The drawback would be the increase of the workload of the containing computer, posing requirement for higher processor efficiency. Also, this configuration would also disable the capability of installation of the back end server behind a firewall, with the front end server (reverse proxy) ahead of the firewall.

Optimisation for flexibility:

- Removal of the restriction of throttle increase. This would add significant risk to the integrity of the gas turbine, especially if it is operated by inexperienced users. It would be an effortless modification that would allow the application to be used in more realistic situations, where harsh testing and rapid power increase may be required. This version of the application should only be addressed to experienced professional gas turbine operators.
- Increase of the time to shut down upon network connection loss. Also, increase of the inactivity limit, which currently was set to 3 minutes without acceptance of any type of commands.
- Formation of a resource API of higher abidance to the RESTful configuration, by distribution of the designed root resources in more than one web applications. This would increase the ease of interaction with other external applications, but the security would not be as efficient as it is currently, where access to all the resources is granted through a centralised authentication approach.

The security of the application could be additionally enhanced with the installation of an Intruder Detection System (IDS) to monitor the servers of the application for unauthorized access and misuse. There are open source IDS systems available, such as Snort, which is based on signature-detection.

12 CONCLUSIONS - RECOMMENDATIONS FOR FURTHER WORK

The final chapter of the Thesis presents the overall conclusions of the project. Additionally, several recommendations were outlined, for future improvement of this application.

12.1 Conclusions

The design took into account the main features of operation and safety of a gas turbine, and combined them with Internet security and performance elements to eventually produce a system that operates successfully and reliably, at least for the short period of time that it has been experimentally operated. With the involvement of common open source Internet design tools, the final application had obtained a very good approximation of real time remote operation of an aero gas turbine (The term approximation was used, due to inevitable network latency existence). The cost of development was maintained in low levels and the final outcome provided an inexpensive tool for aero gas turbine remote operation - due to the Internet accessibility. The system included basic features to ensure physical safety and mitigation of Internet-based risks. It forms a guideline for future extensions, where security can be enhanced and safety will be ensured by more automated functions, replacing currently introduced additional human observers.

12.1.1 Risk mitigation

The software process was an adaptation of the Spiral Model – a combination of iterative development and the systematic approach of the Waterfall model. The process was mainly risk driven and commenced with a very thorough generic risk assessment, with the application of FTA. The analysis identified the most crucial underlying risks, such as the loss of power supply, the network connection disruption and the various Internet threats (Table 26). The combined application of initial FTA and HAZOP provided a thorough identification of the risks, accompanied by specific actions required for mitigation.

Table 26 – Most significant risks and mitigation actions

Risk source	Mitigation action
Loss of power supply	Power stabiliser and monitoring of mains supply – Safe shutdown when power loss detected
Network connection loss	Automated action upon detection of connection disruption
Internet threats	Form-based authentication and encrypted information over SSL, main web application behind firewall via reverse proxy
Hardware malfunction	Visual indications on extended indication panel for engine observer
Misuse	Prevention of abrupt user inputs by server, acknowledgment of command reception by server
Loss of intercommunication or camera image	Automated shutdown if system inactive for more than 3 minutes, 2 observers placed to monitor the system
Junction box failure	Simple and reliable construction of junction box. Even in failure, system easy to switch by interchanging commands wiring to ECU.

The evolution of the software process took place in predefined phases (rounds), where more detailed risk analysis was applied, adapted to the specific needs of each round. The individual round phase applied HAZOP to identify the procedural risks and define mitigation actions or alternative approaches. The procedural risk analysis prevented the occurrence of unnecessary actions and delays during the development, always with alternative options available. Wherever required, there was physical risk assessment also involved, with the application of FMECA, addressing the criticality of the application. This was necessary in the rounds of the software process that included installation or modification of hardware components. Wherever failure rates from the manufacturers were available, they were used for risk score calculation. For the equipment that did not state explicit failure rates, they were assumed to failure rates conforming to MIL – HDBK – 217F. The hardware components that were modified were assumed to have one level of risk higher than that whilst unmodified. After completion of the software process, all the identified mitigation actions had been implemented.

12.1.2 Quality factors

The design has strongly abided to the quality factors outlined by ISO 9126 standard for software product quality. Abidance to the most important factors was outlined herein.

12.1.2.1 Functionality

The final application has covered the majority of the functionality that was planned by the initial version of the functional requirements. Due to time restriction and priority to safety, security and reliability, several functions were suspended for later development. However, the suitability of the final design has satisfied over 90% of the initially required functions. The data recording system was the only major functionality that was not implemented, but this has not affected the overall application and it can be added later on.

The required safety features were enabled, reducing the potential for rapid throttle commands from the user, unattended operation of the gas turbine after the remote client has been disconnected or closed. The throttle cannot be increased more than 20% in a single transition. The main operation panel is only accessible through a root resource retrieved by the user after successful authentication and a POST request. It cannot be accessed directly by entering the URL of the service with a default GET request. Additionally, the URL leading to the path of the operation panel in the server domain is not accessible directly, as a local redirection to the landing page of the InternetGasturbine web application has been configured in the back end server.

The acquisition of the engine output data was highly accurate with deviations less than 1%, mainly due to the sensitivity difference of the EDT and the designed program, as the latter accommodated representation of higher accuracy of the parameters. The fuel flow was the only parameter not captured accurately, due to the lack of reference data. However, the importance at this stage was the caption of the indications and the transmission through the network, which was successful. Accurate fuel flow would be necessary for diagnostics or trend analysis but was not considered a high priority safety parameter for the operation of the engine at the current stage.

Acquired data is presented to the Internet user with very small latency (a few ms), every 0.5 sec. Thus a very good approximation to real time transmission has been obtained with the use of standard Internet technology. A higher pace would not be realistic in cases of very far placed clients or slow connection. The latency was inevitable, due to physical impedance and process of the data, but not crucial, as the system is monitored in real time by the implemented TSS, which detects absolute or relevant exceedance of the critical parameters EGT and RPM, and performs actions automatically if required. Additionally, connection disruption or delay can be detected by the program on the server side and the system will shut down safely.

The most significant undesired event of power supply loss was addressed with two approaches. The first was by connection of a power stabiliser to ensure sufficient time of power supply in case of disruption, in order to safely shut down the engine and the system. The second was by monitoring the supply with the TSS and if loss is detected, the user will be notified. If the warning remains for 3 minutes without any action from the user, the system will shut down automatically.

The application has been designed with the capability to show interoperability with external applications, such as the TURBOMATCH WebEngine. A simple experiment conducted, indicated the necessary approach to obtain cross domain AJAX requests from an external application to the resources of the InternetGasturbine that acquire the engine and operation environment parameters.

In terms of security, the application was configured to run over SSL with a reverse proxy as the front end, allowing the main container to be installed behind an internal framework. It was also suggested to be accessed through the VPN tunnel of the University to provide an additional layer of protection along with the enabled form based authentication. With the servers placed in physically secured areas, the insider attack potential would be reduced significantly.

The elicitation of the requirements has complied with IEEE - 830 - 1998 Standard about Recommended Practice for Software Requirements Specifications. The validation has followed elements of IEEE-1012-2004 Standard for Software Verification and Validation and the validation targets were based on the DO-178B document about Software Considerations in Airborne Systems and Equipment Certification.

12.1.2.2 Reliability

The overall reliability of the basic software components was estimated with a Discrete Time Markov Chain (DTMC) model, at a value at around 0.48. However, the time period for the collection of the experimental data was very short (approximately 100 hrs), hence the result cannot be considered to be objective. Although the system was new with nearly all of the components newly designed, it has shown a satisfactory level of maturity with low frequency of software faults during the testing period. What was most important in this study was the demonstration of a method that would be adequate, if applied after a sufficient period of operation after the completion of the Thesis, to establish the objective reliability of the system. For any fault that occurred, the application presented the expected fault tolerance and recoverability, demonstrated by the performance of the application self refresh sequence, which functioned properly whenever required and within seconds the system had recovered. Also, the engine had safely shut down with all the settings at the predefined default values.

Elements of the project that contributed to the overall reliability were:

- The low cyclomatic complexity (<10) of the modules.
- Not excessive number of lines of code (LOC) of the individual modules, due to modularity.
- Existence of error messages in certain occasions, such as when internal server errors occur, connection loss or system occupied. However, the message system can still accept significant improvements.

12.1.2.3 Usability and efficiency

The application was designed in a simple manner in order to be understandable with enhanced learn-ability. The host web site had a straightforward presentation with all the enclosed pages listed clearly in a centrally placed navigation bar (Figure 133).

Figure 133 – Welcome page and navigation bar

Links to the manufacturer were available and also a complete operation manual of the connected gas turbine could be found in the server domain. The user is guided through the pages and finally, the engine operation panel web page was designed ergonomically with all the elements grouped in three columns: controls, indications, warnings and cautions. The names of all the elements are evident on the page, reducing the required effort to operate the engine, thus enhancing operability. Finally, a very descriptive training page was introduced, which was a replica of the operation panel – without the functionality – with detailed descriptions of each element upon mouse hovering (Figure 134).

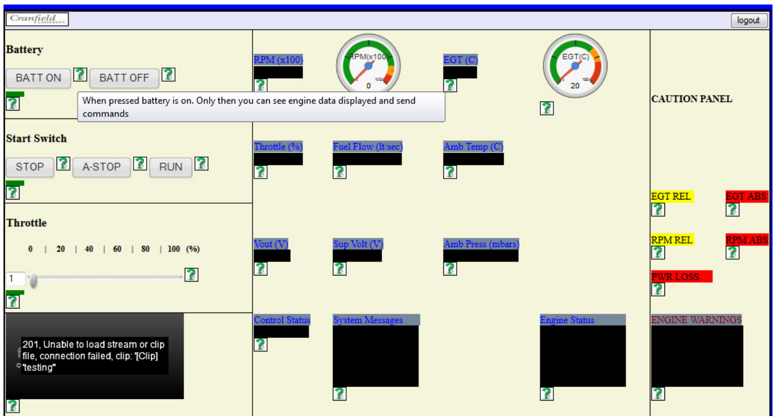


Figure 134 – Training panel

The time for processing the input and output were of the magnitude of a few milliseconds. The start up and refresh times had values less than a second and could be improved if the YUI library for the control elements was installed locally in the server domain, instead of being loaded online. The application thus requires only two external resources to run (YUI library and Google api's – gauges). All the other necessary resources are found locally on the server. Hence the application has performed well up to this point in terms of efficiency.

12.1.2.4 Maintainability and Portability

Maintainability may be expressed in terms of analyse-ability, changeability, stability and testability. The modularity of the application supports the maintainability, as each module can be investigated and debugged individually in order to detect faults. Fault removal or changeability is possible without affecting the other components, as long as the interfaces between them are not altered. After the application had reached the final stage before acceptance validation, it showed general stability to all the applied modifications, as regression testing proved that the components not modified, were not affected by the changes. Additionally, the testability was satisfactory, due to the existence of the General Test Harness, where all the conditions of operation could be simulated and tested. Elements that enhanced maintainability were:

- The low cyclomatic complexity (<10) of the modules.
- Not excessive number of lines of code (LOC) of the individual modules, due to modularity.
- Existence of error messages in certain occasions, such as when internal server errors occur, connection loss or system occupied. However, the message system can still accept significant improvements.
- The access to equipment (engine) manuals and application operation information (training page)

The application could be considered as 50% portable, as the web application related components and MEOCS are adaptable to different operating platforms with reasonable effort. The gas turbine server side components (EIS, ACAS, EISI, TSS) are not adaptable, as they must be installed on the same computer and the LabVIEW components (EIS and ACAS) would not have the full functionality they do in Windows, in a different operating platform.

12.1.3 Reusability

With regards to reusability, the following were concluded:

- The LabVIEW EIS module is tightly couple with a particular gas turbine each time hence not reusable.
- EISI is highly reusable. It requires redesign only of the methods that read the engine output coming out from LabVIEW according to an interface class (Abstract Factory Pattern).
- MEOCS, TSS and InternetGasturbine web application are reusable. The TSS only requires a new input file with the engine nominal values for a different gas turbine.
- The LAN version requires cloning of the user interface accordingly (Prototype Pattern) to accommodate more gas turbines.
- The web version requires a library of user interfaces to accommodate different gas turbines.

The individual components were accompanied with detailed documentation that provided the necessary instructions for addition of more or different gas turbines. All the modules were stored in a repository, available for future access and development.

12.2 Future work

The final section of this chapter presents recommendations for the improvement and further development of the application. These mostly refer to tasks that could be accomplished in the short term and do not involve collaboration with external applications, as described in chapter 10.

Configuration of the reverse proxy to provide cryptography with the use of Network Security Services (NSS), as suggested in section 8.1.4. This action would enhance the security application against the Heartbleed vulnerability of OpenSSL, until the security of the latter is stabilised.

The release of a beta version should be the next step prior to any further developments. This would be the most efficient approach to establish the expected Reliability of the application objectively. With operation under sufficient time, accurate failure rates may be determined and used in the Markov analysis. For a more representative result, the components of the designed DTMC should be analyzed into more simple DTMC models and the expected Reliability of the sub models would produce the individual Reliabilities of the 17 components of the DTMC presented in this Thesis. The reliability analysis can also take into account the randomness of workload or time spent in each component and perform the Monte Carlo analysis introduced by Meedeniya (2011) [94]. Additionally, the beta release would allow the critical functions to be tested for probability of failure on demand (pfd), as proposed in section 9.4, with accomplishment of the required number of runs. The extensive evaluation of expected Reliability would also reveal any bugs that might have not been detected during the relatively short period of testing.

Having the application operational would allow the observation of the software behaviour, when operated from a long distance or even overseas. The observation of the latency would be necessary to adjust the functionality of the client and the sensitivity of the program to the network delays. The frequency of the requests from the client to the server, to acquire the engine and peripheral data, could be dynamically adjusted after periodical checks of the network latency from the client page. The client would require to send periodical requests to a bandwidth tool installed at the server side and the bandwidth value of the connection returned would then be used to adjust the interval of the setInterval() functions of the JavaScript client.

The design and implementation of a Data Recording and Storage System would enhance the functionality, as it would allow interoperability with diagnostics and trend analysis applications. The external applications would then be allowed to interoperate with the current application by acquiring data and processing them accordingly. The operation and fault logs of the connected gas turbine would also be available to the remote user, hence they would provide a thorough insight of the engine and how far in terms of severity the run could reach. The proposal introduces an SQL database installed on the computer connected with the gas turbine. The data that will be received from the EISI module will be provided to a new module that will contain JDBC elements. The data will then be logged into a text file (by calling methods from the FileWriter() class) that will consequently be stored into the database (Figure 135).

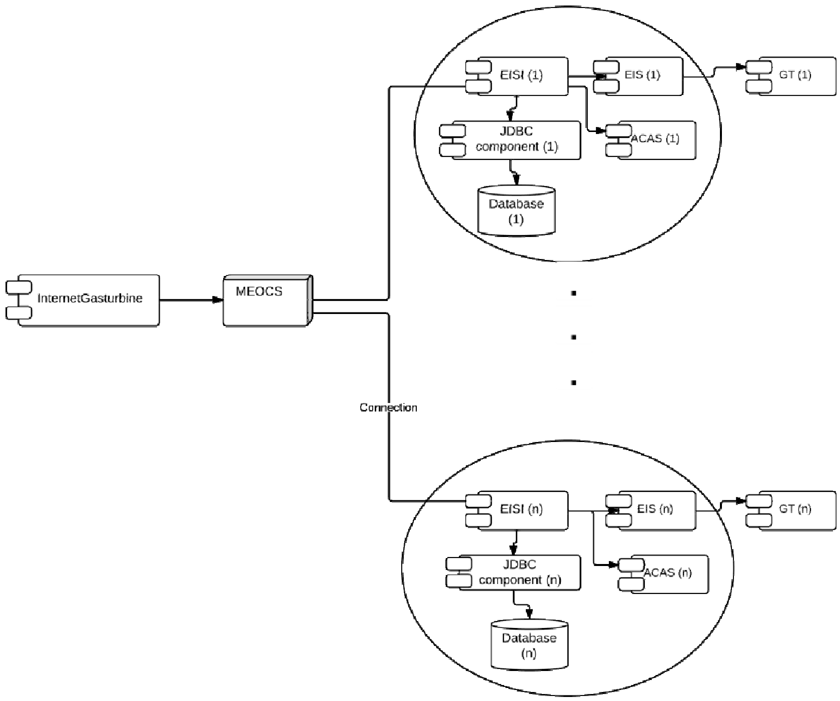


Figure 135 – Data Recording and Storage System

The request for a specific type of engine log will be sent to the appropriate RESTful root service, which will invoke the corresponding method in the MEOCS and from there, connection to the respective Data Recording and Storage system (DRSS) will retrieve the desired file and transfer it via FTP back to the client (Figure 136).

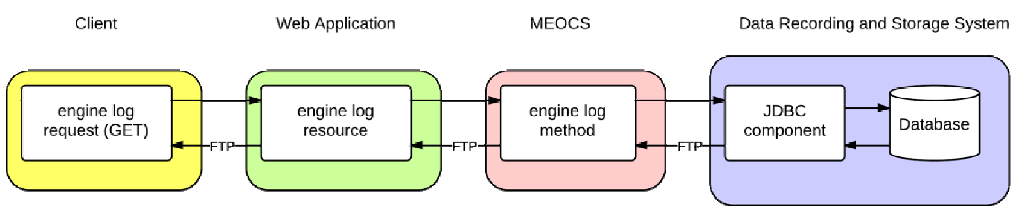


Figure 136 – Request to DRSS

Although the application produces several error messages, there are even more required to cover as many aspects of error generation as possible. Increased error messages would enhance the reliability, the maintainability and the usability of the system. Additional error files should also be created when the operation terminates execution due to network disruption, parameter exceedance or power supply loss. The files generated are suggested to be stored in the local data base, where they will be accessible from the remote user. Another feature that would increase the usability would be the installation of a real time messenger on the operation panel, between the Internet or remote user and the local station PC observer. Although the complexity of the application would increase, the probability of confusion between the user and the observers would be prevented, increasing the safety of the application.

The portability of the application could be improved if the gas turbine side software modules were designed with different programming languages, in order to obtain capability of operation in platforms other than Windows. The National Instruments hardware used for this project could be programmed with MATLAB or C++ instead of LabVIEW. C++ does not require a virtual machine to run on any platform and MATLAB runtime is compatible with Windows, Linux and MAC platforms.

Due to the SSL configuration, the YUI library is loaded online from an SSL friendly library instead of the original YUI library URL. This task has introduced a few milliseconds delay on loading that may cause internal exceptions at the server side modules and terminate the execution. The problem can easily be encountered by installing the YUI library locally in the Glassfish server domain to reduce the time required to load it.

The cost of the application as a whole could be reduced additionally, if the WOWZA media server installed for the live image and sound transmission was replaced with an open source media server. Red5 media server is suggested, which is an open source tool, based on Java (such as WOWZA) and has similar capabilities. Additionally, the definition and latency of the transmitted live image could be improved by replacement of the web cam in use, with an RTSP camera. The operation panel embedded JavaScript Flowplayer is a flash player that requires the shockwave flash plugin to be activated in the browser. The application would become more flexible if the existing HTML5 version of Flowplayer was installed as an option for the user to select, according to the desired browser version capabilities.

In order to increase the automation of the application, the manual battery switch on the engine could also be combined with an electromechanical relay that would operate from a signal generated in LabVIEW (EIS). This feature was currently simulated by a boolean parameter in the program, but it could easily be implemented, provided that the current DAQ unit (NI cDAQ 9174) was replaced by an NI cDAQ 9178, which is an 8 slot chassis and would allow for the installation of the additional NI C series modules required.

The integration with the TURBOMATCH WebEngine is feasible and it would provide a powerful experimental and performance simulation platform for gas turbines. The interoperation can be established with the suggestions presented in section 10.3. Additionally, the capabilities of the application could be expanded by addition of multiple and different gas turbines, as indicated in sections 10.1 and 10.2.

REFERENCES

[1] Alstom (2014), *Plant Support Center for gas turbines*, available at: www.alstom.com/Global/Power/Resources/Documents/Brochures/gas-plant-service-support-center.pdf (accessed 02/24).

[2] Alstom (2014), *Remote monitoring diagnostics services for gas power-plants*, available at: www.alstom.com/Global/Power/Resources/Documents/Brochures/remote-monitoring-diagnostics-services-for-gas-power-plants.pdf (accessed 02/24).

[3] Alstom (2014), *Technical expertise and operational support for gas power plants*, available at: www.alstom.com/power/services/gas/advice-operational-support/ (accessed 04/10).

[4] Ameur, Y., A., Boniol, F. and Wiels, V. (2010), "Toward a wider use of formal methods for aerospace systems design and verification", *International Journal on Software Tools for Technology Transfer,* vol. 12, no. 1, pp. 1-7.

[5] AMT Netherlands (2014), *Olympus AMT Netherlands gas turbine*, available at: http://www.amtjets.com/index.php (accessed 04/10).

[6] AMT Netherlands, (2001), *Olympus Manual*, Geldrop, Netherlands.

[7] Apostolidis, A., Sampath, S., Laskaridis, P. and Singh, R. (2013), "WebEngine - A Web-Based Gas Turbine Performance Simulation Tool", *Proceedings of ASME Turbo Expo 2013 GT2013,* Vol. 4, June 3–7, 2013, San Antonio, Texas, USA, ASME, New York, NY, pp. V004T08A007.

[8] Barnes, S. (2010), *Advanced Software Engineering,* (MSc course notes), Cranfield University, School of Engineering, Cranfield.

[9] Bates, C. (2002), *Web programming: building Internet applications,* 2nd ed, Wiley, Chichester.

[10] Bedford, T. and Cooke, R., M. (2001), *Probabilistic risk analysis: Foundations and methods,* , Cambridge University Press.

[11] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L. and Schnoebelen, P. (2010), *Systems and Software Verification: Model-Checking Techniques and Tools,* 1st ed, Springer Publishing Company, Incorporated.

[12] Blaha, M. and Rumbaugh, J. (2005), *Object-oriented modeling and design with UML,* 2nd edition, Pearson Education, Upper Saddle River, NJ.

[13] Boehm, B. W., Elwell, J. F., Pyster, A. B., Stuckle, E. D. and Williams, R. D. (1982), "The TRW Software Productivity System", *Proceedings of the 6th international conference on Software engineering (ICSE '82),* Los Alamitos, CA, USA, IEEE Computer Society Press, pp. 148.

[14] Boehm, B. W. (1988), "A spiral model of software development and enhancement", *Computer,* vol. 21, no. 5, pp. 61-72.

[15] Boehm, B. W. (1984), "Verifying and Validating Software Requirements and Design Specifications", *Software, IEEE,* vol. 1, no. 1, pp. 75-88.

[16] Bosch, J. (2000), *Design and Use of Software Architecture,* Pearson Education Limited, Harlow, UK.

[17] British Standards Institution, (2010), *BS EN 61508-1:2010 - Functional safety of electrical/electronic/ programmable electronic safety-related systems. General requirements,* British Standards Online.

[18] British Standards Institution, (2007), *BS EN 61025:2007 - Fault Tree Analysis (FTA),* British Standards Online.

[19] British Standards Institution, (2006), *BS EN 60812:2006 - Analysis techniques for system reliability. Procedure for failure mode and effects analysis (FMEA),* British Standards Online.

[20] British Standards Institution, (2001), *BS IEC 61882:2001 - Hazard and operability studies (HAZOP studies). Application guide,* British Standards Online.

[21] Broy, M. (2009), "Seamless Model Driven Systems Engineering Based on Formal Methods", Vol. 9-12 December, pp. 1.

[22] Butler, R. W. and Finelli, G., B. (1993), "The infeasibility of quantifying the reliability of life-critical real-time software", *Software Engineering, IEEE Transactions on,* vol. 19, no. 1, pp. 3-12.

[23] Cadenhead, R. and Lemay, L. (2007), *Sams Teach Yourself Java 6 in 21 days,* Fifth edition, Sams Publishing, USA.

[24] Cangussu, J., W., Haider, S., W., Cooper, K. and Baron, M. (2011), "On the selection of software defect estimation techniques", *Software Testing, Verification and Reliability,* vol. 21, no. 2, pp. 125-152.

[25] Cavada, R., Cimatti, A., Jochim, C., A., Keighren, G., Olivetti, E., Pistore, M. and Tchaltsev, A., (2010), *NuSMV 2.5 User Manual,* FBK-irst, Provo, Italy.

[26] Center for Systems and Software Engineering (2014), *A Spiral Model of Software Development and Enhancement*, available at: http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf (accessed 04/14).

[27] Chaudron, M., Heijstek, W. and Nugroho, A. (2012), "How effective is UML modeling ?", *Software and Systems Modeling,* vol. 11, no. 4, pp. 571-580.

[28] Cimatti, A., Giunchiglia, F., Mongardi, G., Romano, D., Torielli, F. and Traverso, P. (1998), "Model Checking Safety Critical Software with SPIN: An Application to a Railway Interlocking System", in Ehrenberger, W., D. (ed.), *Proceedings of the 17th International Conference on Computer Safety, Reliability and Security (SAFECOMP '98),* Vol. 1516, 5-7 October 1998, Heidelberg Germany, Springer-Verlag, London, pp. 284.

[29] Clarke, P., J., Power, J., F., Babich, D. and King, T., M. (2012), "A testing strategy for abstract classes", *Software Testing, Verification and Reliability,* vol. 22, no. 3, pp. 147-169.

[30] Cofer, D., Whalen, M. and Miller, S., (2008), *Model-Checking of Safety-Critical Software for Avionics*, ERCIM News 75 ed., European Research Consortium for Informatics and Mathematics, Sophia Antipolis Cedex, France.

[31] Di Pietro, R. and Mancini, L., V., (2008), *Intrusion detection systems*, Springer, New York.

[32] Diakostefanis, M., et al. (2009), *Gas turbine leakage fault analysis and detection* (unpublished MSc thesis), Cranfield University, Cranfield, UK.

[33] ECMA International, (2011), *ECMA-262: ECMA Script Language Specification*, 5.1st ed., ECMA International, Geneva, Switzerland.

[34] Elmerabete, J., Hong Li and Rafi, Y. (2010), "Application of Real-Time Control System Using LabVIEW in Distance-Learning", *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on,* Vol. 1, pp. 663.

[35] ESTEREL Technologies (2014), *SCADE Suite*, available at: http://www.esterel-technologies.com/products/scade-suite/ (accessed 05/30).

[36] Everitt, B., S. (2002), *The Cambridge Dictionary of Statistics,* Second edition, Cambridge University Press, Cambridge, UK.

[37] FBK-IRST (2014), *An overview of NuSMV*, available at: http://nusmv.fbk.eu/NuSMV/ (accessed 05/30).

[38] Fedora Directory Server Wiki (2014), *389 Directory Server (Open Source LDAP)*, available at: http://directory.fedoraproject.org/wiki/Mod_nss#What_is_mod_nss.3F (accessed 05/30).

[39] Fielding, R., T. (2014), *REST APIs must be hypertext-driven*, available at: www.roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven (accessed 04/10).

[40] Fielding, R., T. (2000), *Architectural Styles and the Design of Network-based Software Architectures* (Doctoral dissertation thesis), University of California, Irvine.

[41] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (June 1999), *Hypertext Transfer Protocol -- HTTP/1.1,* RFC 2616, Section 5, IETF.

[42] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T. (June 1999), *Hypertext Transfer Protocol -- HTTP/1.1,* RFC 2616, Section 15, IETF.

[43] FloScan Instrument Co. Inc, (2013), *Floscan 201-A6 manual*, USA.

[44] Flowplayer Ltd (2014), *Flowplayer - Video player for the web*, available at: www.flowplayer.org/ (accessed 04/10).

[45] Frappier, M., Fraikin, B., Chossart, R., Chane-Yack-Fa, R. and Ouenzar, M. (2010), "Comparison of model checking tools for information systems", in Song Dong, J. and Zhu, H. (eds.), *Proceedings of the 12th international conference on Formal engineering methods and software engineering (ICFEM'10),* Vol. 6447, 17-19 November 2010, Shangai, China, Springer-Verlag, Berlin, Heidelberg, pp. 581.

[46] Frye, D., W., (2007), *Network security policies and procedures*, Springer, New York.

[47] Fullwood, R. (1999), *Probabilistic Safety Assessment in the Chemical and Nuclear Industries,* Butterworth-Heinemann, UK.

[48] Gallina, B. and Guelfi, N. (2007), "A Template for Requirement Elicitation of Dependable Product Lines", in Sawyer, P., Paech, B. and Heymans, P. (eds.) *Requirements Engineering: Foundation for Software Quality,* Springer Berlin / Heidelberg, , pp. 63-77.

[49] Gamma, E., Helm, R., Johnson, R. and Vissides, J. (1994), *Design Patterns - Elements of Reusable Object - Oriented Software,* Addison - Wesley, USA.

[50] Gargantini, A., and Heitmeyer, C. (1999), "Using Model Checking to Generate Tests from Requirements Specifications", vol. 1687, pp. 146-162.

[51] Getify Solutions (2014), *SON-P: Safer cross-domain Ajax with JSON-P/JSONP*, available at: http://json-p.org/ (accessed 05/30).

[52] github (2014), *Junit*, available at: https://github.com/junit-team/junit (accessed 05/30).

[53] Gogo (2014), *Gogo Inflight Internet - How Gogo works*, available at: www.gogoair.com/gogo/cms/work.do (accessed 05/30).

[54] Gokhale, S. S. and Trivedi, K. S. (2002), "Reliability prediction and sensitivity analysis based on software architecture", *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on,* pp. 64.

[55] Google (2014), *googletest - Google C++ Testing Framework*, available at: http://code.google.com/p/googletest/ (accessed 05/30).

[56] Google (2014), *Java interface to OpenCV and more - Google Project Hosting*, available at: http://code.google.com/p/javacv/ (accessed 05/30).

[57] Google (2014), *Google Code Playground - Gauge*, available at: https://code.google.com/apis/ajax/playground/#gauge (accessed 04/10).

[58] Google (2014), *Visualization: Gauge - Google Charts - Google Developers*, available at: https://developers.google.com/chart/interactive/docs/gallery/gauge#Example (accessed 04/10).

[59] Google (2014), *Red5 Media Server*, available at: www.red5.org (accessed 04/10).

[60] Goševa-Popstojanova, K. and Trivedi, K. S. (2001), "Architecture-based approach to reliability assessment of software systems", *Performance Evaluation; Performance Validation of Software Systems,* vol. 45, no. 2, pp. 179-204.

[61] Graham, S. (2005), *Building Web services with Java: making sense of XML, SOAP, WSDL and UDDI,* 2nd edition, Sams Publishing, Indianapolis, IN.

[62] Gurcan, C. (2007), *Konfiguration eines IntrusionDetection Systems für SPPA-T3000* (unpublished MSc thesis), University of Karlsruhe, Karlsruhe, Germany.

[63] Halsall, F. (1992), *Data communications, computer networks and open systems,* 3rd edition, Addison-Wesley, Wokingham.

[64] Hayes, J., H., Chemannoor, I., R. and Holbrook, E., A. (2011), "Improved code defect detection with fault links", *Software Testing, Verification and Reliability,* vol. 21, no. 4, pp. 299-325.

[65] Hayhurst, K., J., Veerhusen, D., S., Chilenski, J., J. and Rierson, L., K., (2001), *NASA / TM-2001-210 - A Practical Tutorial on Modified Condition/Decision Coverage,* NASA, Hampton, Virginia, USA.

[66] Heffelfinger, D. (2011), *Java EE 6 Development with NetBeans 7,* Packt Publishing, GB.

[67] Herbertsson. (Linkoping University), (2007), *RUT - development handbook 1.3. The Spiral Model v4.0* (unpublished Course handbook), Linkoping, Sweden.

[68] Hignett, K. C. (1996), *Practical safety and reliability assessment,* E & FN Spon, London.

[69] Hilderman, V. and Baghi, T. (2007), *Avionics certification: a complete guide to DO-178 (software), DO-254 (hardware),* Avionics Communications, Leesburg, VA.

[70] Holt, A., (2008), *Network performance analysis*, Springer, London.

[71] IEEE Computer Society, (2009), *Standard IEEE - 1016 - 2009. IEEE Standard for Information Technology—Systems Design—Software Design Descriptions*, IEEE, New York, USA.

[72] IEEE Computer Society, (2005), *IEEE-1012-2004 Standard for Software Verification and Validation*, IEEE, New York, USA.

[73] IEEE Computer Society, (1998), *IEEE - 830 - 1998 Standard: IEEE Recommended Practice for Software Requirements Specifications*, IEEE, NY, USA.

[74] Insam, E. (2003), *TCP/IP Embedded Internet Applications,* 1st ed, Elsevier, Oxford, UK.

[75] Internet Engineering Task Force (IETF) (2014), *The Secure Shell (SSH) Protocol Architecture*, available at: www.ietf.org/rfc/rfc4251.txt (accessed 05/30).

[76] itseez (2014), *OpenCV*, available at: http://opencv.org/ (accessed 02/24).

[77] Java Community Process (2014), *JSR 311: JAX-RS: The Java^{TM} API for RESTful Web Services*, available at: www.jcp.org/en/jsr/detail?id=311 (accessed 04/10).

[78] Jaw, L., C. and Mattingly, J., D. (2009), *Aircraft engine controls: design, system analysis, and health monitoring,* American Institute of Aeronautics and Astronautics, Reston, VA.

[79] JQuery Foundation (2014), *QUnit*, available at: http://qunitjs.com (accessed 04/10).

[80] Kadono, M., Tsuchiya, T. and Kikuno, T. (2009), "Using the NuSMV Model Checker for Test Generation from Statecharts", *Dependable Computing, 2009. PRDC '09. 15th IEEE Pacific Rim International Symposium on,* pp. 37.

[81] Kaikko, J., Talonpoika, T. and Sarkomaa, P. (2002), "Remote on-line monitoring and diagnostics of a power plant", *Proceedings of the Australian Universities Power Engineering Conference (AUPEC) AUPEC2002,* 29 September - 2 October 2002, Melbourne, Australia.

[82] Kemeny, J., G. and Snell, J., L. (1976; 1960), *Finite Markov chains,* Springer-Verlag, New York.

[83] Kim, M., Choi, Y., Kim, Y. and Kim, H. (2008), "Formal Verification of a Flash Memory Device Driver: An Experience Report", vol. 5156, pp. 144-159.

[84] Ko, C. C., Chen, B. M., Jianping Chen, Zhuang, Y. and Chen Tan, K. (2001), "Development of a web-based laboratory for control experiments on a coupled tank apparatus", *Education, IEEE Transactions on,* vol. 44, no. 1, pp. 76-86.

[85] Law, M., A. (2007), *Simulation Modeling & analysis,* Fourth ed, McGraw - Hill, New York, NY, USA.

[86] LED center (2014), *LED series parallel array wizard*, available at: www.led.linear1.org/led.wiz (accessed 04/10).

[87] Li, W. (2005), *Risk assessment of power systems: models, methods, and applications,* Wiley, Hoboken, N.J. , Great Britain.

[88] Li, Y. G. (2002), "Performance-analysis-based gas turbine diagnostics: a review", *Proceedings of the Institution of Mechanical Engineers Part A Journal of Power and Energy,* vol. 216, no. 5, pp. 363.

[89] Littlewood, B., and Wright, D. (1997), "Some conservative stopping rules for the operational testing of safety critical software", *Software Engineering, IEEE Transactions on,* vol. 23, no. 11, pp. 673-683.

[90] Lufthansa Technik (2014), *manage/m Technical Operations Suite*, available at: www.manage-m.com/managem_page_about (accessed 05/30).

[91] MathWorks (2014), *MATLAB*, available at: www.mathworks.co.uk/products/matlab/ (accessed 05/30).

[92] MathWorks (2014), *Simulink Coder*, available at: http://www.mathworks.co.uk/products/simulink-coder/index.html (accessed 05/30).

[93] Meedeniya, I., Moser, I., Aleti, A. and Grunske, L. (2011), "Architecture-based Reliability Evaluation Under Uncertainty", *Proceedings of the Joint ACM SIGSOFT Conference -- QoSA and ACM SIGSOFT Symposium -- ISARCS on Quality of Software Architectures -- QoSA and Architecting Critical Systems -- ISARCS,* Boulder, Colorado, USA, ACM, New York, NY, USA, pp. 85.

[94] Meedeniya, I., Moser, I., Aleti, A. and Grunske, L. (2012), "Evaluating probabilistic models with uncertain model parameters", *Software & Systems Modeling,* , pp. 1-21.

[95] Microsoft Corporation (2014), *How to diagnose a test TCP/IP or NetBios network connections in Windows Server 2003*, available at: http://support.microsoft.com/kb/323388 (accessed 05/30).

[96] Miller, K. W., Morell, L. J., Noonan, R. E., Park, S. K., Nicol, D. M., Murrill, B. W. and Voas, J. M. (1992), "Estimating the probability of failure when testing reveals no failures", *Software Engineering, IEEE Transactions on,* vol. 18, no. 1, pp. 33-43.

[97] Montgomery, D. C. (2008), *Design and analysis of experiments,* 7th edition, Wiley; John Wiley distributor, Hoboken, N.J.; Chichester.

[98] Montgomery, D., C. and Runger, G., C. (1999), *Applied statistics and probability for engineers,* 2nd edition, Wiley, Chichester.

[99] Mozilla Developer Network (2014), *Network Security Services - Mozilla | MDN*, available at: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS (accessed 05/30).

[100] National Instruments (2014), *LabVIEW System Design Software*, available at: http://www.ni.com/labview/ (accessed 05/30).

[101] National Instruments (2014), *NI cDAQ-9174 CompactDAQ 4-Slot USB Chassis*, available at: http://sine.ni.com/nips/cds/view/p/lang/en/nid/207535 (accessed 05/30).

[102] National Instruments (2014), *NI 9263 4-Channel, 100 kS/s, 16-bit, ±10 V, Analog Output Module*, available at: http://sine.ni.com/nips/cds/view/p/lang/en/nid/208806 (accessed 05/30).

[103] National Instruments (2012), *Prove It Works: Using the Unit Test Framework for Software Testing and Validation,* White Paper 8082, Austin, Texas, USA.

[104] National Instruments (2011), *How to Do a Serial Loopback Test,* White paper 3450, Austin, Texas, USA.

[105] National Instruments (2009), *Calling LabVIEW from other programming languages,* White Paper 5719, Austin, Texas, USA.

[106] National Instruments (2003), *LabVIEW User Manual,* , Austin, Texas, USA.

[107] National Instruments Discussion Forums (2014), *Calling LabVIEW from other programming languages*, available at: http://forums.ni.com/t5/LabVIEW-Developers-Feature/Calling-LV-from-other-languages/td-p/324017 (accessed 05/30).

[108] Okajima, H., S., S., Ledel, L., C., Fragnito, H., L. and Rocha, H., V. (2006), "WebLab Development Using a Java and LabView Integrated Solution for Kyatera Network", *3rd TIDIA Fapesp Workshop,* 15 -17 November, Sao Paolo, Brasil, pp. 204.

[109] Oliveira, M. F. S., Redin, R. M., Carro, L., da Cunha Lamb, L. and Wagner, F. R. (2008), "Software Quality Metrics and their Impact on Embedded Software", *Model-based Methodologies for Pervasive and Embedded Software, 2008. MOMPES 2008. 5th International Workshop on,* pp. 68.

[110] OpenBSD (2014), *OpenSSH*, available at: www.openssh.org/ (accessed 05/30).

[111] Oracle, (January 2013), *The Java EE 6 Tutorial - Part No: 821-1841-16*, Redwood City, California, USA.

[112] Oracle (2014), *Introducing the Java EE 6 Platform*, available at: www.oracle.com/technetwork/articles/javaee/javaee6overview-141808.html (accessed 05/30).

[113] Oracle (2014), *JavaServer Faces Tschnology Overview*, available at: www.oracle.com/technetwork/java/javaee/overview-140548.html (accessed 05/30).

[114] Oracle (2014), *Java SE - Documentation*, available at: www.oracle.com/technetwork/java/javase/documentation/index.html (accessed 02/24).

[115] Oracle (2014), *System (Java Platform SE 7)*, available at: http://docs.oracle.com/javase/7/docs/api/java/lang/System.html (accessed 02/24).

[116] Oracle (2014), *Oracle Security Alert for CVE-2013-0422*, available at: www.oracle.com/technetwork/topics/security/alert-cve-2013-0422-1896849.html (accessed 04/10).

[117] Oracle (2014), *Java Servlet Technology Overview*, available at: www.oracle.com/technetwork/java/overview-137084.html (accessed 04/10).

[118] Oracle (2014), *Java Garbage Collection Basics*, available at: www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html (accessed 05/30).

[119] Pautasso, C., Zimmermann, O. and Leymann, F. (2008), "Restful Web Services vs. Big' Web Services: Making the Right Architectural Decision", *Proceedings of the 17th International Conference on World Wide Web,* Beijing, China, ACM, New York, NY, USA, pp. 805.

[120] Pilidis, P. and Palmer, J. P. (2008), *Gas Turbine Theory and Performance,* (MSc course notes), Cranfield University, School of Engineering, Cranfield.

[121] Poore, J. H., Mills, H. D. and Mutchler, D. (1993), "Planning and certifying software system reliability", *Software, IEEE,* vol. 10, no. 1, pp. 88-99.

[122] predic8 GmbH (2014), *Introduction into REST Web Services*, available at: http://predic8.com/rest-webservices.htm (accessed 05/30).

[123] Pressman, R. S. (2010), *Software engineering: a practitioner's approach,* 7th ed, McGraw-Hill Higher Education, New York.

[124] Qiao, Y., Liu, G.,P., Zheng, G. and Hu, W. (2010), "NCSLab: A Web-Based Global-Scale Control Laboratory With Rich Interactive Features", *Industrial Electronics, IEEE Transactions on,* vol. 57, no. 10, pp. 3253-3265.

[125] Qin, Z., Zheng, X. and Xing, J. (2008), "Introduction to Software Architecture", in *Software Architecture,* Springer Berlin Heidelberg, pp. 1-33.

[126] Qualtech Consulting (2014), *Summary of Difference Between DO-178B and DO-178C*, available at: www.faaconsultants.com/html/do-178c.html (accessed 05/30).

[127] Richardson, L. and Ruby, S. (2007), *Restful Web Services,* First edition, O'Reilly Media, Sebastopol, California, USA.

[128] Roy, A., Kim, D., S. and Trivedi, K., S. (2012), "Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees", *Security and Communication Networks,* vol. 5, no. 8, pp. 929-943.

[129] Saravanamuttoo, H. I. H. (2008), *Gas turbine theory,* 6th edition, Pearson Prentice Hall, Upper Saddle River, N.J.

[130] Siemens (2014), *The Benchmark in Controls – Technical Highlights Siemens Power Plant Automation™ – SPPA-T3000*, available at: www.energy.siemens.com/hq/pool/hq/automation/automation-control-pg/sppa-t3000/T3-B-ContrSys-us-V11.pdf (accessed 05/30).

[131] Software Engineering institute (2014), *Capability Maturity Model Integration (CMMI)*, available at: www.sei.cmu.edu/cmmi/ (accessed 05/30).

[132] Sommerville, I. (2006), *Software engineering,* 8th edition, Pearson Education, GB.

[133] SourceForge (2014), *PHP/Java Bridge*, available at: www.php-java-bridge.sourceforge.net/doc/installation.php (accessed 04/10).

[134] Stamatelatos, M., (2002), *Fault Tree Handbook with Aerospace Applications*, http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf ed., NASA, Washington DC, USA.

[135] The Apache Software Foundation (2014), *mod_proxy - Apache HTTP Server*, available at: http://httpd.apache.org/docs/2.2/mod/mod_proxy.html (accessed 02/24).

[136] The Apache Software Foundation (2014), *Apache Tomcat*, available at: http://tomcat.apache.org (accessed 04/10).

[137] The OpenSSL Project (2014), *OpenSSL Security Advisory*, available at: www.openssl.org/news/secadv_20140407.txt (accessed 04/30).

[138] The PHP Group (2014), *PHP: Hypertext Preprocessor*, available at: www.php.net (accessed 04/10).

[139] Trivedi, K. S. (1982), *Probability and statistics with reliability, queuing, and computer science applications,* Prentice-Hall, Englewood Cliffs, NJ.

[140] US Department of Defence, (1991), *MIL-HDBK-217F: Reliability Prediction of Electronic Equipment*, Washington DC, USA.

[141] Vacca, J., R., (2007), *Practical Internet security*, Springer, New York, NY.

[142] Vardi, M., Y. (2001), "Branching vs. Linear Time: Final Showdown", in Margaria, T. and Yi, W. (eds.), *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001),* Vol. 20031, 2-6 April 2001, Genova, Italy, Springer-Verlag, London, UK, pp. 1.

[143] W3C Working Group Note, 11 February 2004 (2014), *Web Services Architecture*, available at: www.w3.org/TR/2004/NOTE-ws-arch-20040211/#whatis (accessed 04/10).

[144] Wade, J. B., Fujinoki, H., Coffman, A., Feerer, D. M. and Hauck, A. G. (2010), "A cross-layer approach for mitigating denial of service attacks: Device-driver packet filter and remote firewalling", *International Journal of Communication Networks and Information Security,* vol. 2, no. 3, pp. 231-239.

[145] Walsh, P., P., Fletcher, P. and Knovel, (2004), *Gas turbine performance*, 2nd ed., Blackwell Science, Malden, MA.

[146] Wang, C., Hachtel, G., D. and Somenzi, F., (2006), *Abstraction refinement for large scale model checking*, Springer, New York.

[147] Watson, A., H. and McCabe, T., J. (September 1996), *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric,* NIST Special Publication 500-235, National Institute of Standards and Technology, Gaithersburg, Maryland, USA.

[148] Wikipedia (2014), *Heartbleed*, available at: http://en.wikipedia.org/wiki/Heartbleed (accessed 04/30).

[149] Wilding, E. (2003), "Password Disclosure Matrix", *Computer Fraud & Security,* vol. 2003, no. 7, pp. 4-5.

[150] World Wide Web Consortium (2014), *Hypertext Transfer Protocol* , available at: www.w3.org/Protocols/ (accessed 04/10).

[151] World Wide Web Consortium (2014), *Javascript Web APIS,* available at: www.w3.org/standards/webdesign/script (accessed 20/11).

[152] World Wide Web Consortium (2014), *HTTP Methods: GET vs. POST*, available at: www.w3schools.com/tags/ref_httpmethods.asp (accessed 02/24).

[153] World Wide Web Consortium (2014), *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition)*, available at: www.w3.org/TR/xhtml1 (accessed 04/10).

[154] World Wide Web Consortium (2014), *HTML 4.01 Specification*, available at: www.w3.org/TR/html401 (accessed 04/10).

[155] World Wide Web Consortium (2014), *HTML5*, available at: www.w3.org/TR/html5 (accessed 04/10).

[156] World Wide Web Consortium (2014), *Cascading Style Sheets*, available at: www.w3.org/Style/CSS (accessed 04/10).

[157] World Wide Web Consortium (2014), *JavaScript Tutorial*, available at: www.w3schools.com/js/DEFAULT.asp (accessed 04/10).

[158] Wowza Media Systems (2014), *Wowza Streaming Server*, available at: www.wowza.com (accessed 04/10).

[159] Yahoo (2014), *YUI library*, available at: http://yuilibrary.com/ (accessed 05/30).

[160] Yahoo (2014), *Button - YUI Library*, available at: http://yuilibrary.com/yui/docs/button (accessed 04/10).

[161] Yahoo (2014), *Slider - YUI Library*, available at: http://yuilibrary.com/yui/docs/slider (accessed 04/10).

[162] Yoo, S. and Harman, M. (2012), "Test data regeneration: generating new test data from existing test data", *Software Testing, Verification and Reliability,* vol. 22, no. 3, pp. 171-201.

[163] You, I., Lenzini, G., Ogiela, M., R. and Bertino, E. (2012), "Defending against insider threats and internal data leakage", *Security and Communication Networks,* vol. 5, no. 8, pp. 831-833.

APPENDICES

Appendix A Functional Requirements

This is an excerpt from the System Requirements Specification Document, produced in accordance with IEEE Std 830 – 1998(R2009), 'IEEE Recommended Practice for Software Requirements Specifications'. Due to the length of the SRS, only the User's Requirements and the analysis of the Functional Requirements section were included herein. The former were distinguished in Functional and Non – functional. The latter were presented in structured natural language.

A.1 User's Requirements

A.1.1 Functional Requirements

1. The system shall enable the start and total operation of the gas turbine from:

   - A nearby physically connected PC station

     And/or

   - A remote user through Internet connection or LAN

The control shall allow rapid and precise selection of the throttle settings. Throttle shall advance in increments of 1% ranging from 0 to 100%. However variation of throttle position shall not be totally uncontrollable. Excessively fast rate above 20% total range/sec should be prohibited.

Rationale: The structure of the system will allow operation of the engine either from the PC station or from a remote PC through the Internet. The flexibility of the throttle control is required in order to obtain the whole range of operation in which the engine would be subjected if manually controlled.

2. It shall Record and log the total time of operation of the gas turbine and also record the time of operation above 90% RPM.

Rationale: Operating time recording and logging is crucial for the maintenance and the life of the engine and it is integrated to the operation procedure.

3. The input and output of the software is shown in the following table:

Engine Input (Software Output)	Engine Output (Software input)
Throttle angle Start/Shut down command	**<u>Measurements</u>** Exhaust Gas Temperature Shaft Speed RPM Fuel Flow (Not via ECU) **<u>Condition</u>** Throttle angle Fuel Pump Voltage Status of switch ECU supply voltage ECU errors

* Additionally ambient temperature and pressure will be acquired for utilization within the Engine Operation Control Software

* Electrical and fuel supply are controlled by the Automatic Start Unit through the ECU, hence we do not need to intervene. Also, electrical connection and fuel tank supply inevitably must be applied manually.

4. The system shall monitor the Electronic Control Unit (ECU) errors and allow the operator to take appropriate actions. The errors are:

   - Start – up sequence error or low RPM error
   - Switch input channel failed
   - Regulator (throttle) input channel failed
   - Exhaust temperature error
   - High RPM error (RPM > 118,000)
   - pump battery empty

Rationale: The error indications of the ECU notify the operator of the actual state of the embedded engine control system and inform him to quit the operation.

5. The system shall monitor the engine RPM, Exhaust Gas Temperature (EGT), the fuel pump voltage and the fuel flow and provide the readings to the user clearly on the screen. Any exceedance of operation limits shall be visually highlighted and allow the remote operator to take proper action. Also, the remote user or the PC station user will have the ability to apply instant emergency stop of the engine operation from his/her station. If there is no response from the user within 5 seconds the software will proceed to the AUTOSTOP selection of the switch and shut down. The limits are:

- RPM < 26,000 RPM or > 118,000 RPM

- EGT < 300 ^{o}C or > 850 ^{o}C

Rationale: It is indisputably necessary for the remote user to have a real time and plausible depiction of the engine operation parameters. It is a major issue of safety when something is not operating properly and required action needs to be taken immediately.

6. Whenever RPM, EGT or fuel flow exceed predefined normally expected values throughout the throttle angle range, the software shall notify the user and if no response, idle the engine for 3 minutes and then proceed to shut down. However, once the user responds, he will be able to continue operating the engine. Predefined values to be confirmed (2 standard deviations of mean measured value at each throttle setting position). Values to be determined experimentally

Rationale: This is an automation feature which not only will notify the user if an abnormal (but not hazardous) condition exists, but will also protect from further deterioration in case of lack of notice from the user.

7. It shall enable the recording of the measurement readings of the engine for selected increments of time on demand of the user. These are RPM, EGT and fuel flow. The recorded data shall be maintained and provided to related software systems which will conduct Diagnostics or Trend Analysis of the engine.

Rationale: Diagnostics and Trend Analysis are very essential applications for the assurance of proper maintenance, safe operation and long life of the gas turbine. Automatic creation of the required data files will allow the maintenance staff to conduct the procedures with more ease and accuracy.

8. The system shall enable intercommunication between the users and also provide live picture of the gas turbine test house to the remote user during operation.

Rationale: Safety during the remote operation of the engine will be enhanced when the participating individuals can have, a common clear picture and the ability to communicate and co-ordinate their actions.

9. The system by default shall shut down (AUTOSTOP mode) the gas turbine in the event of electric power or connection disruption. The electric supply voltage must be monitored by appropriate hardware installation and in case of instability detection the system shall proceed as stated before. If there is no response from the user for 3 minutes, the engine will shut down.

Rationale: Automatic termination at the occurrence of power or connection disruption will prevent any confusion between the participating individuals, who while attempting to determine and evaluate the outstanding problem, may lose tracking of the engine operation and become prone to applying erroneous or contradicting control orders.

A.1.2 Non – Functional Requirements

10. The system shall comprise of reliable, robust software and stable network connection.

Rationale: The whole project may be considered as a critical system, thus requires critical software design procedures. Robustness will allow for accuracy in the control of the gas turbine via the net. Stability and reliability will assure constant control of the behavior of the engine with the least probabilities of hazard occurrence.

11. Manual controllability of the engine shall be available during remote operation at any time.

Rationale: This feature will prevent undesirable conditions should any loss of network control occurs. In any case, the control will be continued manually before the situation becomes irreversible.

12. The software shall be secure enough to prevent undesirable access and malicious alterations.

Rationale: Any exposure of the software could lead not only to improper function of the system, but also to hazardous events. Therefore, securing the software and the data transfer is a matter of major concern.

13. The system shall operate strictly with the presence of 2 individuals, when operating from the nearby PC station, or 3 when operating remotely through the Internet. One shall be in the area of the engine in the test house, another in the PC station, and finally, one at a remote PC, if Internet or LAN operation has been selected. The system shall not allow the operation to commence unless it is assured that all 2 (or 3) members of Staff are in position.

Rationale: Human presence in every level of control will provide the capability to maintain control of the engine from a lower level, if connection with a higher level is lost. In the worst case, the presence of a member of Staff in the physical area of the engine test house will be able to take over the controls if total connection with the other 2 levels occurs.

14. The system must be able to perform on Windows XP onwards and LINUX.

Rationale: These are the main OS of the PC's on University campus.

15. The designed software must comprise of easily decoupled components in order to obtain a reusable core that can communicate with various interfaces, connecting other gas turbine types and collaborating with external applications for performance simulation or diagnostics.

Rationale: The application must have provisions for generic applications, as it will be expanded in the future to include other gas turbines as well and also allow integration with the TURBOMATCH WebEngine Internet application for remote gas turbine performance simulation.

A.2 Functional requirements analysis - Mode 0: Web Application

Pre-conditions: Web servers ready

A.2.1 Functional Requirement 0.1: Authentication

SYSTEM						
User Input				Response		
User name - Password				Enter main page		
Web Application						
No.	Method	Input	Parameter	Type	Condition/Action	Response
1.	User Authenication	User name	<UserName>	String	Boolean – TRUE	Show Introduction Page
		Password	<Password>	String	Boolean – TRUE	
2.	Authenication Failure	User name | Password	<UserName> & <Password>	String	Boolean - FALSE	Show Login Error Page
3.	Program Execution	Link selection	<run>	http Request	xmlHttp – call service	Run Program – Show Operation Panel
4.	Data Reception	Link selection	<read>	http Request	xmlHttp – call service (repeat every 0.5 sec)	Display Default Values
5.	Logout	Link selection	<logout>	http Request	xmlHttp – call service	Close Operation Panel - Show Logout Page
6.	Server Error	Server → Browser: Engine Data	<DataInput>	String	if<DataInput>not correct	Close Operation Panel - Show Error Page
7.	Connection Loss	Browser → Server: Data Reception	<TimeBetweenRequests>	Integer	If <TimeBetweenRequests> > 7	Close Operation Panel - Show Error Page
8.	Deny Multiple Users	Link selection	<unique_user>	http Request	xmlHttp – <unique_user> != TRUE	Show Return Page

A.2.2 Functional requirement 0.2: System monitoring

SYSTEM						
User Input				Response		
Camera → ON				Show live video		
Engine Operation Control Software						
No.	Method	Input	Parameter	Type	Condition/Action	Response
1.	Live video	Camera Stream: ON	<liveImage>	Boolean	If TRUE	Display video

A.3 Functional requirements analysis - Mode1: Engine start – No malfunction

Pre-conditions: No engine problems during start
Mechanical connections successful
Authentication successful

A.3.1 Functional requirement 1.1: Engine Start

Client and Web application						
No.	Method	User Input	Parameter	Type	Condition/Action	Response
1.	Start Request	Start Switch → ON	<Command> <Flag>	xmlHttp request	If <Flag>=2 & <Command> = 3	Engine Running at Idle Display: Throttle Position EGT RPM Fuel Pump Voltage
Engine Operation Control Software						
No.	Method	Input from User	Parameter	Type	Condition/Action	Response
2.	Start Command	Start Switch → ON	<Switch>	Integer (1 – 3)	If <Switch> = 3	(<Switch>=3) → Engine Interface Software
Engine Interface Software						
No.	Method	Input	Parameter	Type	Condition/Action	Response
3.	Generate Start Signal	Start Switch → ON	<Switch>	Integer (1 – 3)	If <Switch> = 3	Analog dc signal 5V → ECU port input C Analog dc signal 0V → ECU port input D
4.	RPM Acquisition	ECU serial output port B: data Normal Set Data byte 2	<RPM>	Double	Engine Idle	<RPM> → EOCS
5.	EGT Acquisition	ECU serial output port B data: Normal Set Data byte 3	<EGT>	Double	Engine Idle	<EGT> → EOCS
6.	Throttle Position Acquisition	ECU serial output port B data: Normal Set Data byte 4	<ThrottlePos>	Integer	Engine Idle	< ThrottlePos > → EOCS
7.	Vout Acquisition	ECU serial output port B data: Normal Set Data byte 5	<Vout>	Double	Engine Idle	<Vout> → EOCS
8.	Fuel Flow Acquisition	Flow meter transmitter: Open collector transistor output	<FuelFlow>	Double	Engine Idle	<FuelFlow> → EOCS

9.	Engine Status Information	ECU serial output port B data: Normal Set Data byte 3 – bit1, bit2	<Engine Status>	String	Engine Idle	<Engine Status> → EOCS
10.	Time Count	Time Counter	<TimeOperating>	Double	<TimeOperating> = 0	<TimeOperating> = +<TimeOperating> → EOCS
Engine Operation Control Software						
No.	Method	Input	Parameter	Type	Condition/Action	Response
11.	RPM Transmission	Parameter from EIS	<RPM>	Double	Engine Idle	<RPM> → EOCS
12.	EGT Transmission	Parameter from EIS	<EGT>	Double	Engine Idle	<EGT> → EOCS
13.	Throttle Position Transmission	Parameter from EIS	<ThrottlePos>	Integer	Engine Idle	<ThrottlePos>→ EOCS
14.	Vout Transmission	Parameter from EIS	<Vout>	Double	Engine Idle	<Vout>→ EOCS
14.	Fuel Flow Transmission	Parameter from EIS	<FuelFlow>	Double	Engine Idle	<FuelFlow>→ EOCS
15.	Engine Status Information Transmission	Parameter from EIS	<Engine Status>	String	Engine Idle	<Engine Status> → EOCS
16.	Time Logging	Parameter from EIS	<TimeOperating>	Double	Refreshed every second	Log in database
Client and Web application						
No.	Method	Input	Parameter	Type	Condition/Action	Response
17.	RPM Display	Parameter from EOCS	<RPM>	Double	Engine Idle	Display on Operation Panel Indicator
18.	EGT Display	Parameter from EOCS	<EGT>	Double	Engine Idle	Display on Operation Panel Indicator
19.	Throttle Position Display	Parameter from EOCS	<ThrottlePos>	Integer	Engine Idle	Display on Operation Panel Indicator
20.	Vout Display	Parameter from EOCS	<Vout>	Double	Engine Idle	Display on Operation Panel Indicator
21.	Fuel Flow Display	Parameter from EOCS	<FuelFlow>	Double	Engine Idle	Display on Operation Panel Indicator
22.	Engine Status Information Display	Parameter from EOCS	<Engine Status>	String	Engine Idle	Engine Status Panel: Started – Idle Calibrated

A.4 Functional requirements analysis - Mode2: Engine start – Malfunction present

Pre-conditions: No engine problems during start
Mechanical connections successful
Authentication and Configuration Identical as previous mode

A.4.1 Functional requirement 2.1 – Engine Start

Client and Web Application						
No.	Method	User Input	Parameter	Type	Condition/Action	Response
1.	Start Request	Start Switch → ON	<Command> <Flag>	xmlHttp request	If <Flag>=2 & <Command> = 3 ECU error present	Engine Does not Start Display: Error Messages
Engine Operation Control Software						
No.	Method	Input from User	Parameter	Type	Condition/Action	Response
2.	Error Start Command	Start Switch → ON	<Switch>	Integer (1 – 3)	If <Switch> = 3	(<Switch>=3) → Engine Interface Software
Engine Interface Software						
No.	Method	Input	Parameter	Type	Condition	Response
3.	Error Start Signal Generation	Start Switch → ON	<Switch>	Integer (1 – 3)	If <Switch> =3	Analog dc signal 5V → ECU port input C Analog dc signal 0V → ECU port input D
4.	Start Error 1 Generation	ECU serial output port B data: Error Set Data byte 2 – bit1	<StartError1>	String	Engine Not Started	<StartError1> = switch channel not present → EOCS
5.	Start Error 2 Generation	ECU serial output port B data: Error Set Data byte 2 – bit2	<StartError2>	String	Engine Not Started	<StartError2> = throttle channel not present → EOCS
6.	Start Error 3 Generation	ECU serial output port B data: Error Set Data byte 2 – bit5	<StartError3>	String	Engine Not Started	<StartError3> = supply low error → EOCS
7.	Engine Status Information	ECU serial output port B data: Normal Set Data byte 3 – bit1, bit2	<Engine Status>	String	Engine Not Started	<Engine Status> → EOCS
Engine Operation Control Software						
No.	Method	Input	Parameter	Type	Condition/Action	Response
8.	Transmit Start Error 1	Parameter from EIS	<StartError1>	String	Engine Not Started	<StartError1> = switch channel not present → Web App
9.	Transmit Start Error 2	Parameter from	<StartError2>	String	Engine Not Started	<StartError2> = throttle channel not present →

		EIS				Web App
10.	Transmit Start Error 3	Parameter from EIS	<StartError3>	String	Engine Not Started	<StartError3> = supply low error → Web App
11.	Engine Status Information Transmission	Parameter from EIS	<Engine Status>	String	Engine Not Started	<Engine Status> → EOCS
Client and Web Application						
No.	Method	Input	Parameter	Type	Condition/Action	Response
12.	Display Start Error 1	Parameter from EOCS	<StartError1>	String	Engine Not Started	Engine Warning Panel: switch channel not present
13.	Display Start Error 2	Parameter from EOCS	<StartError2>	String	Engine Not Started	Engine Warning Panel: throttle channel not present
14.	Display Start Error 3	Parameter from EOCS	<StartError3>	String	Engine Not Started	Engine Warning Panel: supply low error
15.	Engine Status Information Display	Parameter from EOCS	<Engine Status>	String	Engine Not Started	Engine Status Panel: Switch on/auto

A.5 Functional requirements analysis - Mode3: Engine running

Pre-conditions: Engine has started up successfully

A.5.1 Functional Requirement 3.1: Acceleration

Client and Web Application						
No	Method	Input from User	Parameter	Type	Condition/Action	Response
1.	Proper Throttle Advance	Throttle Lever → FWD	<Command> <Flag>	xmlHttp request	If <Flag>=3 & {<Command> (final) – <Command> (previous) < 20}	(Throttle = n, n ∈ {0,100}) → EOCS
2.	Rapid Throttle Advance	Throttle Lever → FWD	<Command> <Flag>	xmlHttp request	If <Flag>=3 & {<Command> (final) – <Command> (previous) > 20}	(Throttle = n, n ∈ {0,100}) → EOCS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
3.	Throttle Increase Proper	Parameter from Web App	<Throttle>	Int [100] {For (i=0, i<100, i++) Throttle[i] =i}	If (Throttle[final] – Throttle[previous])<20	(Throttle = n, n ∈ {0,100}) → EIS
4.	Throttle Increase Rapid				If (Throttle[final] – Throttle[previous])>20	Error message → Web App

Engine Interface Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
5.	Throttle Increase Signal Generation	From ECS: Throttle Lever → FWD	<Throttle>	Int [100] {For (i=0, i<100, i++) Throttle [i] =i)	If (Throttle[final] – Throttle[previous])<20	Analog dc signal 0 - 5V → ECU port input D (∀ Throttle[i]++, analog dc signal +0.05V)
6.	RPM Acquisition	ECU serial output port B data: Normal Set Data byte 2	<RPM>	Double	Engine Idle	<RPM> → EOCS
7.	EGT Acquisition	ECU serial output port B data: Normal Set Data byte 3	<EGT>	Double	Engine Idle	<EGT> → EOCS
8.	Throttle Position Acquisition	ECU serial output port B data: Normal Set Data byte 4	<ThrottlePos>	Integer	Engine Idle	< ThrottlePos > → EOCS
9.	Vout Acquisition	ECU serial output port B data: Normal Set Data byte 5	<Vout>	Double	Engine Idle	< Vout > → EOCS
10.	Fuel Flow Acquisition	Flow meter transmitter: Open Collector signal	<FuelFlow>	Double	Engine Idle	< FuelFlow > → EOCS
11.	Time Count	Time Counter	<TimeOperating>	Double	<TimeOperating> = last value	<TimeOperating> = +<TimeOperating> → EOCS
12.	Time Count High RPM	Time Counter Above 100,000 RPM	<TimeOperatingAbove>	Double	<TimeOperatingAbove> =0	<TimeOperatingAbove> = +<TimeOperating> → EOCS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
13.	RPM Transmission	Parameter from EIS	<RPM>	Double	Engine Stabilised after acceleration	<RPM> → Web App
14.	EGT Transmission	Parameter from EIS	<EGT>	Double	Engine Stabilised after acceleration	<EGT> → Web App
15.	Throttle Position Transmission	Parameter from EIS	<ThrottlePos>	Integer	Engine Stabilised after acceleration	< ThrottlePos > → Web App
16.	Vout Transmission	Parameter from EIS	<Vout>	Double	Engine Stabilised after acceleration	< Vout > → Web App
17.	Fuel Flow Transmission	Parameter from EIS	<FuelFlow>	Double	Engine Stabilised after acceleration	< FuelFlow > → Web App
18.	Time Logging	Parameter from EIS	<TimeOperating>	Double	Refreshed every second	Logged in database
19.	Time Logging High RPM	Parameter from EIS	<TimeOperatingAbove>	Double	Refreshed every second	Logged in database
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
20.	RPM Display	Parameter from EOCS	<RPM>	Double	Engine Stabilised after acceleration	Display on Browser Indicator – Refreshed every 0.5sec
21.	EGT Display	Parameter from EOCS	<EGT>	Double	Engine Stabilised after acceleration	Display on Browser Indicator – Refreshed every 0.5sec

22.	Throttle Position Display	Parameter from EOCS	<ThrottlePos>	Integer	Engine Stabilised after acceleration	Display on Browser Indicator – Refreshed every 0.5sec
23.	Vout Display	Parameter from EOCS	<Vout>	Double	Engine Stabilised after acceleration	Display on Browser Indicator – Refreshed every 0.5sec
24.	Fuel Flow Display	Parameter from EOCS	<FuelFlow>	Double	Engine Stabilised after acceleration	Display on Browser Indicator – Refreshed every 0.5sec
25	Rapid Throttle Notification	Parameter from EOCS	<Rapid Throttle>	String	Engine has not changed state	Display on Browser Indicator

A.5.2 Functional Requirement 3.2: Deceleration

Pre-conditions: Engine has been previously accelerated and stabilized successfully

Client and Web Application						
No	Method	Input from User	Parameter	Type	Condition/Action	Response
1.	Throttle Decrease	Throttle Lever → RWD	<Command> <Flag>	xmlHttp request	If <Flag>=3 }	(<Command> = n, n ∈ {0,100}) → EOCS
Engine Operation Control Software						
No	Method	Input from User	Parameter	Type	Condition/Action	Response
2.	Throttle Decrease Proper	Throttle Lever → RWD	<Throttle>	Int [100] {For (i=0, i<100, i++) Throttle[i] =i}	Transmit <Throttle>	(<Throttle> = n, n ∈ {0,100}) → EIS
Engine Interface Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
3.	Throttle Decrease Signal Generation	From ECS: Throttle Lever → RWD	<Throttle>	Int [100] {For (i=0, i<100, i++) Throttle[i] =i}	Generate signal	Analog dc signal 0 - 5V → ECU port input D (∀ Throttle[i]--, signal -0.05V)
4.	RPM Acquisition	ECU serial output port B data: Normal Set Data byte 2	<RPM>	Double	Engine Stabilised after decceleration	Double (RPM, EGT, Vout, FuelFlow) and Integer (ThrottlePos) → Engine Operation Control Software
5.	EGT Acquisition	ECU serial output port B data: Normal Set Data byte 3	<EGT>	Double	Engine Stabilised after decceleration	<RPM> → EOCS
6.	Throttle Position Acquisition	ECU serial output port B data: Normal Set Data byte 4	<ThrottlePos>	Integer	Engine Stabilised after decceleration	<EGT> → EOCS

7.	Vout Acquisition	ECU serial output port B data: Normal Set Data byte 5	<Vout>	Double	Engine Stabilised after decceleration	< ThrottlePos > → EOCS
8.	Fuel Flow Acquisition	Flow meter transmitter: Open Collector signal	<FuelFlow>	Double	Engine Stabilised after decceleration	< Vout > → EOCS
9.	Time Count	Time Counter input	<TimeOperating>	Double	<TimeOperating> = last value	<FuelFlow > → EOCS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
10.	RPM Display	Parameter from EIS	<RPM>	Double	Engine Stabilised after deceleration	Display on Client Screen – Refreshed every 25ms
11.	EGT Display	Parameter from EIS	<EGT>	Double	Engine Stabilised after deceleration	Display on Client Screen – Refreshed every 25ms
12.	Throttle Position Display	Parameter from EIS	<ThrottlePos>	Integer	Engine Stabilised after deceleration	Display on Client Screen – Refreshed every 25ms
13.	Vout Display	Parameter from EIS	<Vout>	Double	Engine Stabilised after deceleration	Display on Client Screen – Refreshed every 25ms
14.	Fuel Flow Display	Parameter from EIS	<FuelFlow>	Double	Engine Stabilised after deceleration	Display on Client Screen – Refreshed every 25ms
15.	Time Logging	Parameter from EIS	<TimeOperating>	Double	Refreshed every second	Logged in database
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
16.	RPM Display	Parameter from EOCS	<RPM>	Double	Engine Stabilised after decceleration	Display on Browser Indicator – Refreshed every 0.5sec
17.	EGT Display	Parameter from EOCS	<EGT>	Double	Engine Stabilised after decceleration	Display on Browser Indicator – Refreshed every 0.5sec
18.	Throttle Position Display	Parameter from EOCS	<ThrottlePos>	Integer	Engine Stabilised after decceleration	Display on Browser Indicator – Refreshed every 0.5sec
19.	Vout Display	Parameter from EOCS	<Vout>	Double	Engine Stabilised after decceleration	Display on Browser Indicator – Refreshed every 0.5sec
20.	Fuel Flow Display	Parameter from EOCS	<FuelFlow>	Double	Engine Stabilised after decceleration	Display on Browser Indicator – Refreshed every 0.5sec

A.5.3 Functional Requirement 3.3: Parameter Logging

Pre-conditions: Engine is running successfully

SYSTEM						
User Input				Response		
Record				Record parameter – save in Data Storage System		
End Recording				Stop Recording parameters		
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
1.	Record Stable	Command: Record	<Record>	String	If RPM = Const for t ≥ 30 sec	Record <RPM>, <EGT>, <FuelFlow>, <Tambient>, <Pambient> → .txt file to Data Storage System
2.	Record Unstable	Command: Record	<Record>	String	If RPM ≠ Const	Stop Recording – Display message
3.	End Recording	Command: End Recording	<EndRecording>	String	If Recording in process	Stop Recording – Display message

A.6 Functional requirements analysis - Mode4: Trouble Shooting

A.6.1 Functional requirement 4.1: Protection when absolute operating limits exceeded

Preconditions: Engine running with no abnormalities previously detected

SYSTEM						
Input				Response		
Engine parameters				If parameters exceed absolute values display warning message and shutdown		
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
1.	RPM To TSS	From EIS: RPM	<RPM>	Double	Constantly	- <RPM> → TSS, every 25ms - Refreshed every 25ms
2.	EGT To TSS	From EIS: EGT	<EGT>	Double	Constantly	- <EGT> → TSS, every 25ms - Refreshed every 25ms
Trouble Shooting Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
3.	RPM Absolute Check	From EOCS: RPM	<RPM>	Double	If 30,000< RPM <118,000	Boolean <RPM absolute Warning> True → EOCS Integer <Switch> =2→ EIS
4.	EGT Absolute Check	From EOCS: EGT	<EGT>	Double	If RPM >30000 & EGT>850	Boolean <EGT absolute Warning> True → EOCS Integer <Switch> =2→ EIS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
5.	RPM Warning Generation	From TSS: Boolean Parameters	<RPM absolute Warning>	Boolean	If TRUE	Boolean <RPM absolute Warning> True → Web App

6.	EGT Warning Generation	From TSS: Boolean Parameters	<EGT absolute Warning>	Boolean	If TRUE	Boolean <EGT absolute Warning> True → Web App
Engine Interface Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
7.	Emergency Shutdown	Start Switch → AUTO	<Switch>	Integer (1 – 3)	<Switch> =2	Analog DC signal 2.5V → ECU port input C
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
8.	RPM Absolute Display	From EOCS: Boolean Parameters	<RPM absolute Warning>	Boolean	If TRUE	- Display RPM absolute warning on browser indicator - Shut down engine and program
9.	EGT Absolute Display	From EOCS: Boolean Parameters	<EGT absolute Warning>	Boolean	If TRUE	- Display EGT absolute warning on browser indicator - Shut down engine and program

A.6.2 Functional requirement 4.2: Protection when relative limits exceeded

Preconditions: Engine running with no abnormalities previously detected
Temperature measured in degrees Centigrade
Pressure measured in Pascals
Fuel Flow measured in Kgr/sec

SYSTEM	
Input	Response
Engine parameters	If parameters exceed relative values display warning message and if no event in 3 minutes then shutdown

Ambient Conditions Acquisition System (ACAS)						
No	Method	Input	Parameter	Type	Condition/Action	Response
1.	Ambient Pressure Acquisition	Serial Input RS232 from sensor	<AmbPressure>	Double	Refreshed every 60 sec	<AmbPressure> → TSS
2.	Ambient Temperature Acquisition	Digital Input from thermocouple	<AmbTemp>	Double	Refreshed every 60 sec	<AmbTemp> → TSS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
3.	RPM To TSS	From EIS: RPM	<RPM>	Double	If <Throttle> = const for t = 5 sec	- <RPM> → TSS, every 25ms - Refreshed every 25ms
4.	EGT To TSS	From EIS: EGT	<EGT>	Double	If <Throttle> = const for t = 30 sec	- <EGT> → TSS, every 25ms - Refreshed every 25ms

5.	Throttle Position to TSS	From EIS: Throttle Position	<Throttle>	Integer	<Throttle> = const for t = 5 sec	- <Throttle> → TSS, every 25ms - Refreshed every 25ms
Trouble Shooting Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
6.	Ambient Pressure Deviation	From ACAS: Ambient Temperature	<AmbPressure>	Double	- <δ> = <AmbPressure>/101,325 - Recalculated every 60 sec	<AmbPressure>, <δ> → EOCS
7.	Ambient Temperature Deviation	From ACAS: Ambient Pressure	<AmbTemp>	Double	-< θ> = <AmbTemp>/15 - Recalculated every 60 sec	<AmbTemp>, <θ> → EOCS
8.	Throttle Position Check	From EOCS: Throttle Position	<Throttle >	Integer	For (i=0, i<50, i++), Check <Throttle[i]>	Determine <Throttle[i]>
9.	RPM Correction and Check	From ECS: RPM	<RPM>	Double	- <Throttle[i]> = Known - <RPMc> = <RPM>/<√θ> - If <RPMc> > {RPM(mean at <Throttle[i]>) +2σ}	Boolean <RPM relative warning> = True → EOCS
10.	EGT Correction and Check	From EOCS: EGT	<EGT>	Double	-<Throttle[i]> = Known - <EGTc> = <EGT>/<θ> - If <EGTc> > {EGT(mean at <Throttle[i]>) +2σ}	Boolean <EGT relative warning> = True → EOCS
11.	Shutdown Engine when relative exceedance present	Relative Warnings and Elapsed Time	<RPM relative warning> <EGT relative warning> <Elapsed Time>	Boolean Boolean Integer	If (<RPM relative warning> | <EGT relative warning>) = TRUE & (<Elapsed Time>) ≥180 sec	Integer <Switch> = 2 → EIS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
12.	RPM Caution Transmission	From TSS: Boolean Parameter	<RPM relative warning>	Boolean	If TRUE	-Display Warning message - Log in Database - As per FR 5.2
13.	EGT Caution Transmission	From TSS: Boolean Parameter	<EGT relative warning>	Boolean	If TRUE	-Display Warning message - Log in Database - As per FR 5.2
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
14.	Ambient Pressure Display	From EOCS: Double Parameter	<Ambient Pressure>	Double	Refresh every 0.5 sec	Display Ambient Pressure indication on browser
15.	Ambient Temperature Display	From EOCS: Double Parameter	<Ambient Temperature>	Double	Refresh every 0.5 sec	Display Ambient Temperature indication on browser
16.	RPM Caution Display	From EOCS: Boolean Parameter	<RPM relative warning>	Boolean	If TRUE	Display RPM relative warning indication on browser
17.	EGT Caution Display	From EOCS: Boolean Parameter	<EGT relative warning>	Boolean	If TRUE	Display EGT relative warning indication on browser

A.6.3 Functional requirement 4.3: Local Protection

Preconditions: No other exceedance present.

SYSTEM						
Input				**Response**		
External power value Network Connection condition				If either parameters fall below a predefined value give respective warnings. If no event in 30 seconds shutdown		
Main Supply Interface System (MSIS)						
No	**Method**	**Input**	**Parameter**	**Type**	**Condition/Action**	**Response**
1.	Power Supply Voltage Acquisition	Analog AC Signal	<Voltage>	Double	Refreshed every 25ms	<Voltage> → TSS
Engine Interface System (EIS)						
2.	Physical command signal loss	NI9263 signals	<Command_signal_loss>	Boolean	Refreshed constantly	Boolean <Manual_Warning> = True, indication on Observer Panel and Double <warning_light> = 9 volts → EDT - indicators 3 & 4 illuminated
3.	Physical signal loss	NI DAQ 9174 signal	<Signal_loss>	Boolean	Refreshed constantly	Boolean <Manual_Warning> = True, indication on Observer Panel & EDT indicators 1 & 2 cease to illuminate
4.	Engine output loss	Serial Input	<Engine_input>	Boolean	Refreshed constantly	Boolean <Manual_Warning> = True, after 10 sec: Double <Switch> = 2.5
5.	Observer Operation Control	Input from Observer	<Observer_switch>	Boolean	<Observer_switch> = TRUE	String <Operation status> = "Observer" → EOCS
Trouble Shooting Software						
No	**Method**	**Input**	**Parameter**	**Type**	**Condition/Action**	**Response**
6.	Power Supply Voltage to TSS	From MSIS	< Mains_loss >	Boolean	If (< Mains_loss >) < 1 Volt for t>3 sec Or <Voltage> > (TBD) Refreshed every 25ms	Boolean < Mains_loss > : TRUE → EOCS
7.	Power Supply Emergency shutdown	Parameter From TSS & <time>	<Mains_loss > <time>	Boolean Integer	(<Mains_loss >) = TRUE & (<time>) ≥ 180 sec	- <Throttle> = 0 → EIS - <Switch>=2 → EIS
Engine Operation Control Software						
No	**Method**	**Input**	**Parameter**	**Type**	**Condition/Action**	**Response**
8.	Power Supply Warning Transmission	From TSS: Boolean Parameter	< Mains_loss >	Boolean	If TRUE	< Mains_loss > → Web App
9.	Observer Operation Control Transmission	Input from EIS	<Observer status>	String	Transmit constantly	String <Operation status> = "Observer" → Web APP
Client and Web Application						
No	**Method**	**Input**	**Parameter**	**Type**	**Condition/Action**	**Response**

10.	Power Supply Warning Display	From EOCS: Boolean Parameter	< Mains_loss >	Boolean	If TRUE	Display Warning message on browser indicator
11.	Observer Operation Control Display	Input from EOCS	<Observer status>	String	Update every 500 ms	Display on browser indicator

A.7 Functional requirements analysis - Mode5: Engine Shutdown

A.7.1 Functional requirement 5.1: Normal Shutdown

Preconditions: No abnormalities observed.

Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
1.	Normal Auto Stop	From User: Start Switch: AUTO OFF	<Switch>	Integer (1 – 3)	<Switch> = 2	(<Switch>=2) → EOCS Software
2.	Normal Off	From User: Start Switch: OFF	<Switch>	Integer (1 – 3)	<Switch> = 1	(<Switch>=1) → EOCS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
3.	Normal Auto Stop	From Web App	<Switch>	Integer (1 – 3)	<Switch> = 2	(<Switch>=2) → Engine Interface Software
4.	Normal Off	From Web App	<Switch>	Integer (1 – 3)	<Switch> = 1	(<Switch>=1) → Engine Interface Software
Engine Interface Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
5.	Auto Off Signal Generation	Start Switch → AUTO OFF	<Switch>	Integer (1 – 3)	<Switch> =2	Analog dc signal 2.5V→ ECU port input C
6.	Off Signal Generation	Start Switch → OFF	<Switch>	Integer (1 – 3)	<Switch> =1	Analog dc signal 0V → ECU port input C
7.	Time Count Completion	Time Counter	<TimeOperating>	Double	<TimeOperating> = last value	<TimeOperating> = +<TimeOperating> → ECS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
8.	Shutdown RPM Transmission	Parameter from EIS	<RPM>	Double	<RPM> = 0	<RPM> → Web App
9.	Shutdown EGT Transmission	Parameter from EIS	<EGT>	Double	<EGT> = reducing	<EGT> → Web App
10.	Shutdown Throttle Position Transmission	Parameter from EIS	<ThrottlePos>	Integer	<ThrottlePos> = 0	< ThrottlePos > → Web App
11.	Shutdown Vout Transmission	Parameter from EIS	<Vout>	Double	<Vout> = 0	< Vout > → Web App
12.	Shutdown Fuel Flow Transmission	Parameter from EIS	<FuelFlow>	Double	<FuelFlow> = 0	< FuelFlow > → Web App

13.	Total Time Operating Transmission	Parameter from EIS	<TimeOperating>	Double	<TimeOperating> = last value	Log in Database
14.	Total Time High RPM Operating Transmission	Parameter from EIS	<TimeOperatingAbove>	Double	<TimeOperatingAbove> = last value	Log in Database
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
15.	Shutdown RPM Display	Parameter from EOCS	<RPM>	Double	<RPM> = 0	Display on browser indicator
16.	Shutdown EGT Display	Parameter from EOCS	<EGT>	Double	<EGT> = reducing	Display on browser indicator
17.	Shutdown Throttle Position Display	Parameter from EOCS	<ThrottlePos>	Integer	<ThrottlePos> = 0	Display on browser indicator
18.	Shutdown Vout Display	Parameter from EOCS	<Vout>	Double	<Vout> = 0	Display on browser indicator
19.	Shutdown Fuel Flow Display	Parameter from EOCS	<FuelFlow>	Double	<FuelFlow> = 0	Display on browser indicator

A.7.2 Functional requirement 5.2: Emergency Shutdown local

Preconditions: Engine output lost or manual control warning on

Engine Interface Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
1.	Emergency Shutdown by observer	Emergency shutdown switched pressed by observer	<Emergency_switch >	Boolean	<Emergency_switch > = TRUE	Analog dc signal 0V → ECU port input D Analog dc signal 2.5V → ECU port input C
2.	Programmatic Shutdown When engine output lost	Serial input USB port 1	<Input>	String	<Input> = null	Analog dc signal 0V → ECU port input D After t=30 ms: 2.5V → ECU port input C
3.	Manual Shutdown	Manual warning signal	<EDT indicators>	Boolean	<EDT indicators> = TRUE	- Junction box switched to manual by on site user - Analog dc signal 2.5V → EIS - <Operation condition > = manual → EOCS
4.	Time Count Completion	Time Counter	<TimeOperating>	Double	<TimeOperating> = last value	<TimeOperating> = +<TimeOperating> → ECS
Engine Operation Control Software						
No	Method	Input	Parameter	Type	Condition/Action	Response
5.	Manual Operation Notification Transmission	Parameter from EIS	<Operation condition >	String	<Operation status > = “manual”	<Operation status > → Web App

6.	Shutdown RPM transmission	Parameter from EIS	<RPM>	Double	<RPM> =0	<RPM> → Web App
7.	Shutdown EGT transmission	Parameter from EIS	<EGT>	Double	<EGT> = reducing	<EGT> → Web App
8.	Shutdown Throttle Position transmission	Parameter from EIS	<ThrottlePos>	Integer	<ThrottlePos> = 0	< ThrottlePos > → Web App
9.	Shutdown Vout transmission	Parameter from EIS	<Vout>	Double	<Vout> = 0	< Vout > → Web App
10.	Shutdown Fuel Flow transmission	Parameter from EIS	<FuelFlow>	Double	<FuelFlow> = 0	< FuelFlow > → Web App
11.	Total Time Operating Log	Parameter from EIS	<TimeOperating>	Double	<TimeOperating> = last value	Log in Database
12.	Total Time High RPM Operating Log	Parameter from EIS	<TimeOperatingAbove>	Double	<TimeOperatingAbove> = last value	Log in Database
Client and Web Application						
No	Method	Input	Parameter	Type	Condition/Action	Response
13.	Manual Operation Notification display	Parameter from EOCS	<Operation status >	String	<Operation status > = “manual”	Display on browser indicator
14.	Shutdown RPM display	Parameter from EOCS	<RPM>	Double	<RPM> =0	Display on browser indicator
15.	Shutdown EGT display	Parameter from EOCS	<EGT>	Double	<EGT> = reducing	Display on browser indicator
16.	Shutdown Throttle Position display	Parameter from EOCS	<ThrottlePos>	Integer	<ThrottlePos> = 0	Display on browser indicator
17.	Shutdown Vout display	Parameter from EOCS	<Vout>	Double	<Vout> = 0	Display on browser indicator
18.	Shutdown Fuel Flow display	Parameter from EOCS	<FuelFlow>	Double	<FuelFlow> = 0	Display on browser indicator

Appendix B Risk Assessment

B.1 Fault Tree Analysis for primary design stage of the system

B.1.1 Introduction

Fault Tree analysis is method for qualitative and quantitative risk assessment of industrial applications, even at early stage design. It is implemented in power plants, chemical, nuclear and aviation industry. The procedure described here is in line with MIL – STD – 882D, Standard Practice for Safety System. It is one of multiple methods used.

B.1.2 Scope

This analysis intends to present a general overview of the system's underlying risks and identify top undesired events along with the related individual events which can cause them. It will initially be a qualitative analysis and an attempt to quantify the results will be done prior to commence of more detailed design of the system. The purpose of the present analysis is to identify potential risks and determine ways of mitigating them at the phase of System Requirements Specification process.

B.1.3 Considerations

The analysis is based on a structure of a system consisting of hardware and software components, as indicated in the related context model and operated by three users at different positions during the operation. It also contains human events and shows their interaction with the system in order to lead to a top event through 'and' or 'or' gates. It is not a detailed risk assessment for individual components or systems of the integrated project.

The result of the quantitative FTA may be either a probability or a rate value. However, the probability itself can be considered as a kind of rate as well, as it expresses the likelihood of an event happening in a defined amount of time during the life cycle of a system or component. In this case, the system will not be in constant operation hence rate estimation would be a more realistic approach to the risk assessment. The rate may be the cycles of operation.

Quantitative FTA would not be easy at this stage due to lack of statistical sample of failure modes and events, as the project is yet in top level design phase and specific components had not yet been decided.

The FTA of the under design project has followed a risk assessment of the Olympus gas turbine operating regularly in the test house, which has been based on the experience of the staff of the test lab after a vast number of operation cycles and has revealed acceptable values of risk for that process. Hereon, major undesired top events are identified and added to the risk of the regular process of operation of the engine manually in the test lab.

B.1.4 Top Events

Two events have been identified as the worst case scenario during an Internet based operation of the engine. After the FTA for each event, a respective follow on event tree is shown.

1. **Total loss of command**. In this situation, the engine runs without the capability of being controlled, neither through the Internet, nor manually. In that case, the only way of stopping it is to shut off the fuel supply manually, once the situation is realized.
2. **Erroneous command order**. The engine does not follow an intended command but instead executes at power settings other than the desired. During this event, the manual control capability is considered to be available and the restoration of the desired power settings can be achieved once the situation is realized and the on – site user takes over manual control.

B.1.5 Individual Events

This section presents the individual events that if combined may lead to the 2 top undesirable events. These were then considered in the FTA followed by an Event Tree Analysis (ETS), presenting the potential consequences of the top events.

<u>Total Loss of Command</u>

M. Loss of power supply
N. Network connection loss
O. Exploit attack
P. Denial of Service (DoS) attack
Q. User Misuse
R. Undetected bugs triggered
S. Manual control switch failure
T. Connection – control hardware malfunction
U. Intercom system failure
V. Camera system failure
W. Malware

X. Inside attack

Erroneous command order

A. User Misuse
B. Undetected bugs triggered
C. Connection – control hardware malfunction
D. Engine Sensor Failure
E. Exploit Attack
F. DoS Attack
G. Intercom system failure
H. Camera system failure
I. Malware
J. Inside Attack

Total loss of Control – Fault Tree

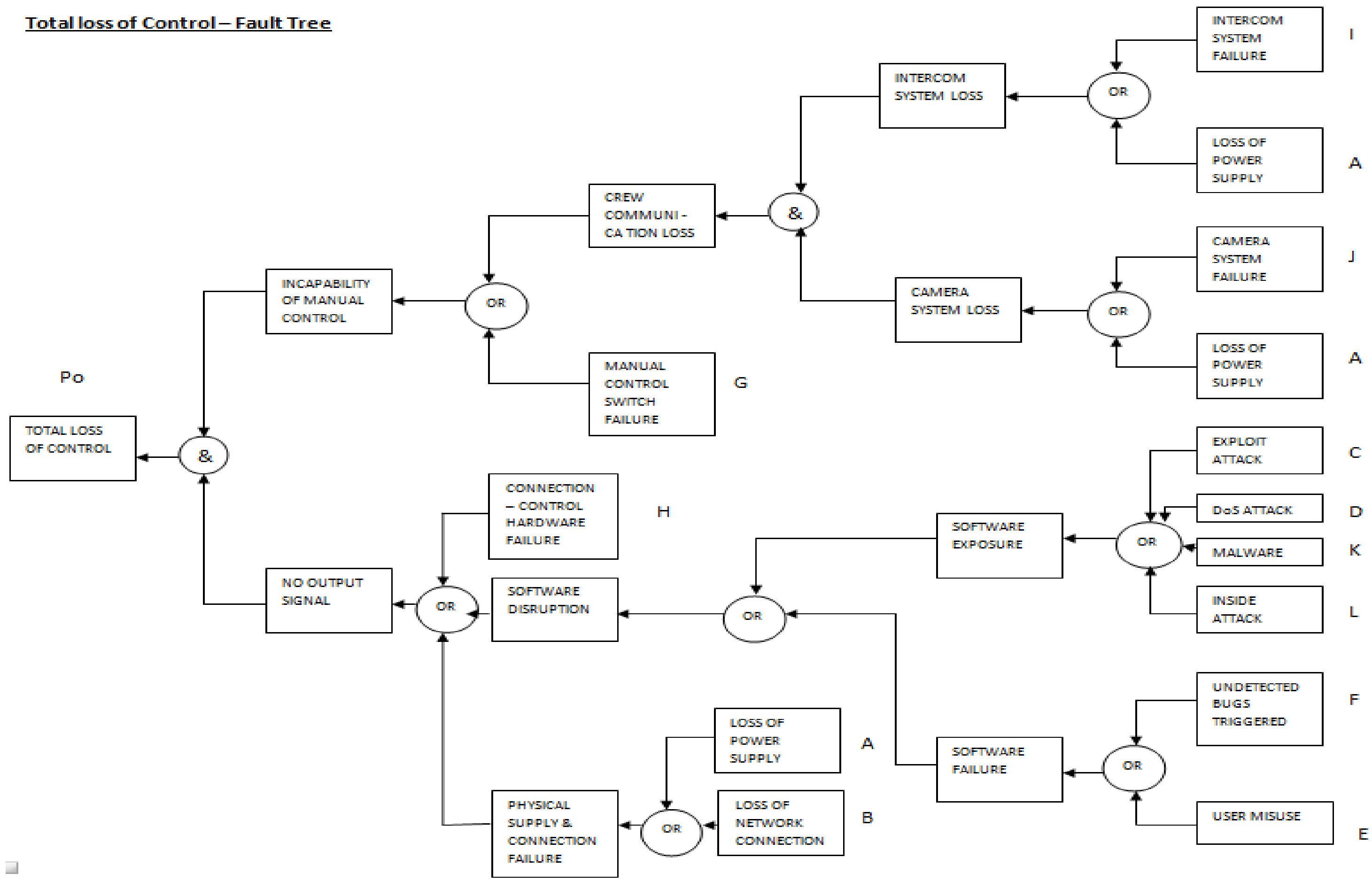


Figure B-1 FTA for Total loss of control event

Total loss of Control – Event Tree

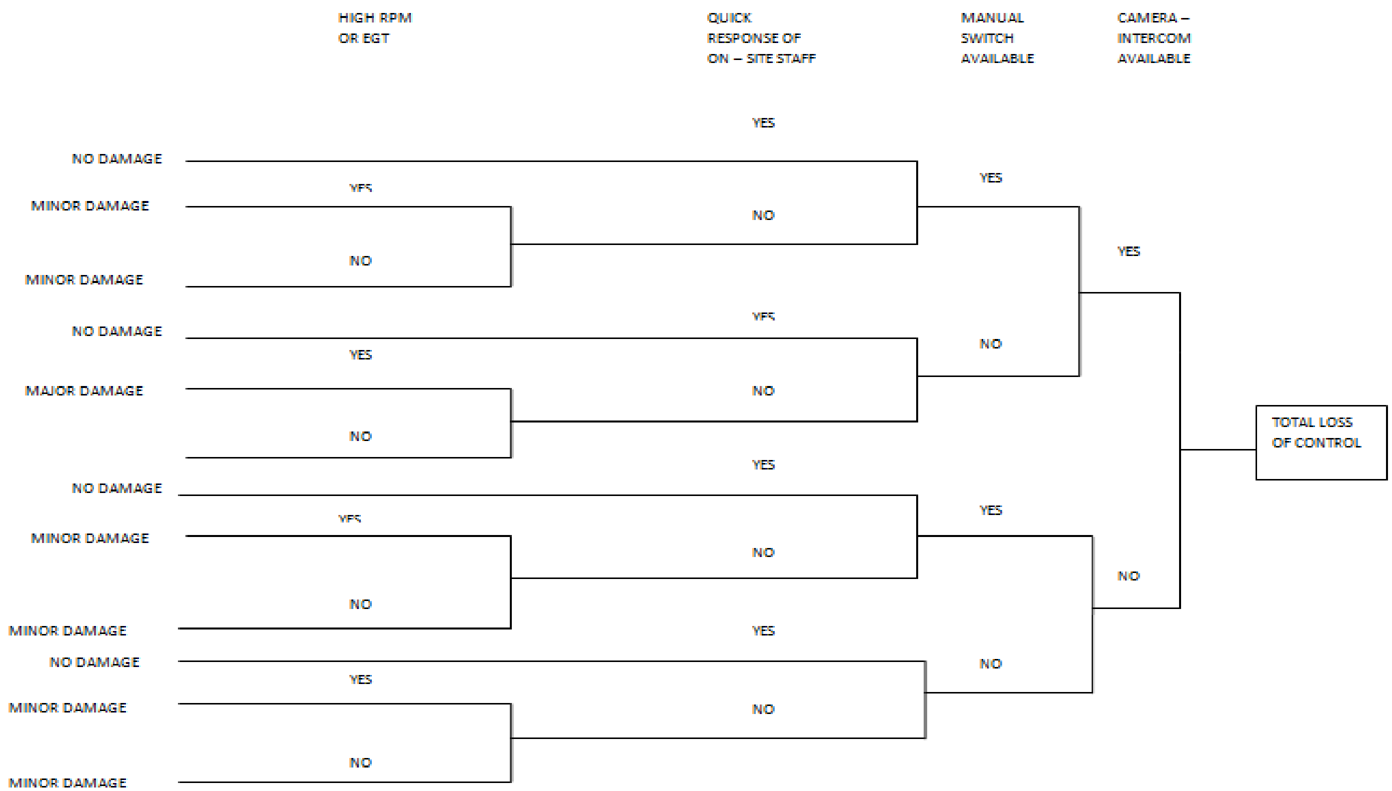


Figure B-2 - ETA for Total loss of control event

Total Control Loss – Boolean analysis with Minimal Cut Sets

P_o: Top undesirable event

P_o = [G+(I+A)(J+A)][A+B+C+D+E+F+K+L+H] =

= (G+IJ+IA+JA+A)(A+B+C+D+E+F+K+L+H) =

= (G+IJ+A)(A+B+C+D+E+F+K+L+H) =

= G(A+B+C+D+E+F+K+L+H)+IJ(A+B+C+D+E+F+K+L+H)+A(A+B+C+D+E+F+K+L+H) =

= A+AG+GB+GC+GD+GE+GF+GK+GL+GH+IJA+IJB+IJC+IJD+IJE+IJF+IJK+IJL+IJH =

= A+GB+GC+GD+GE+GF+GK+GL+GH+IJB+IJC+IJD+IJE+IJF+IJK+IJL+IJH

Observations:

- Large number of combinations: 17 minimal cut – sets
- Event A is most dominant – Can contribute to top event just by itself
- Events I, J and G is also in several sets (8)

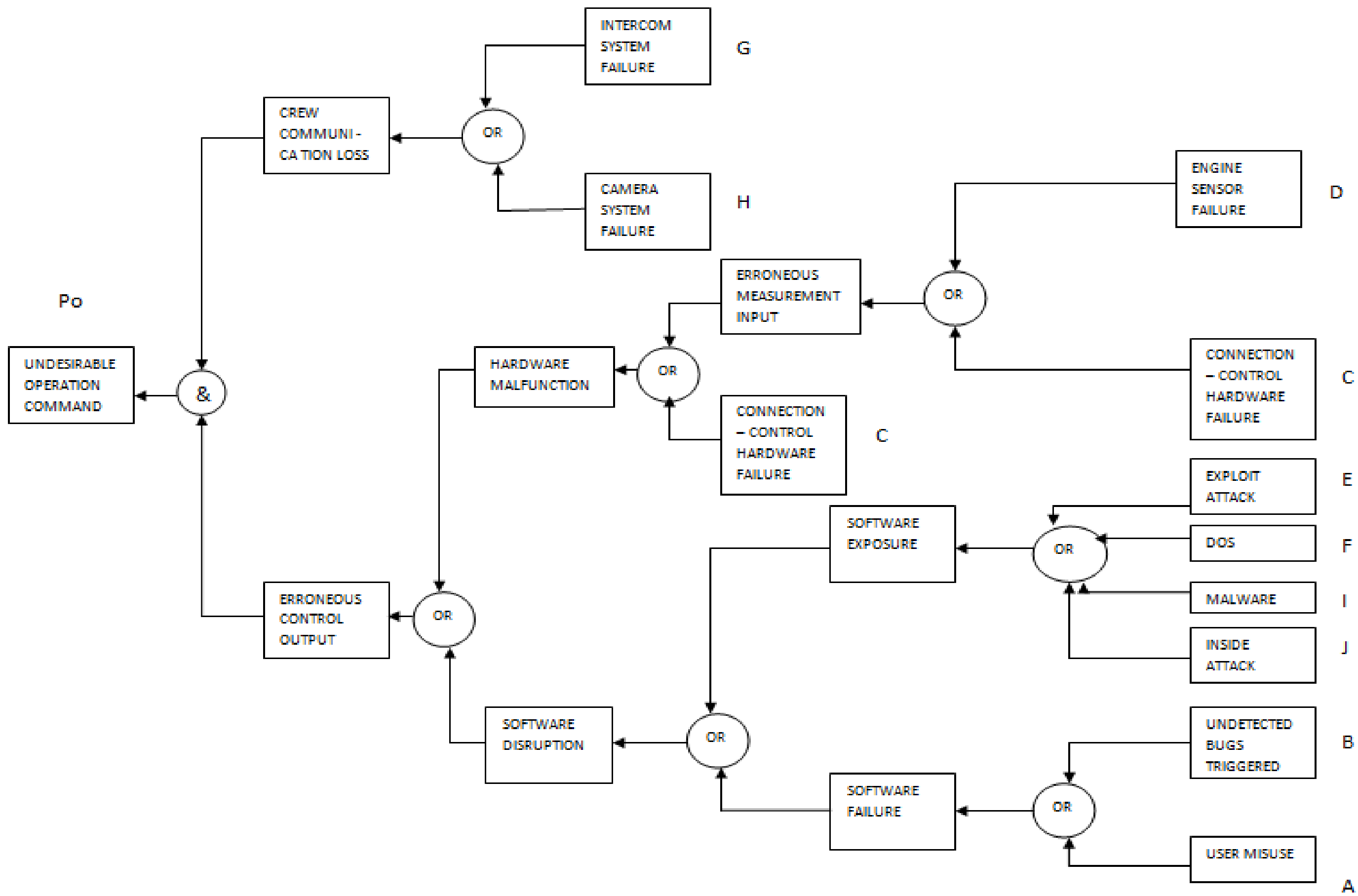


Figure B-3 - FTA for Undesirable Operation Command event

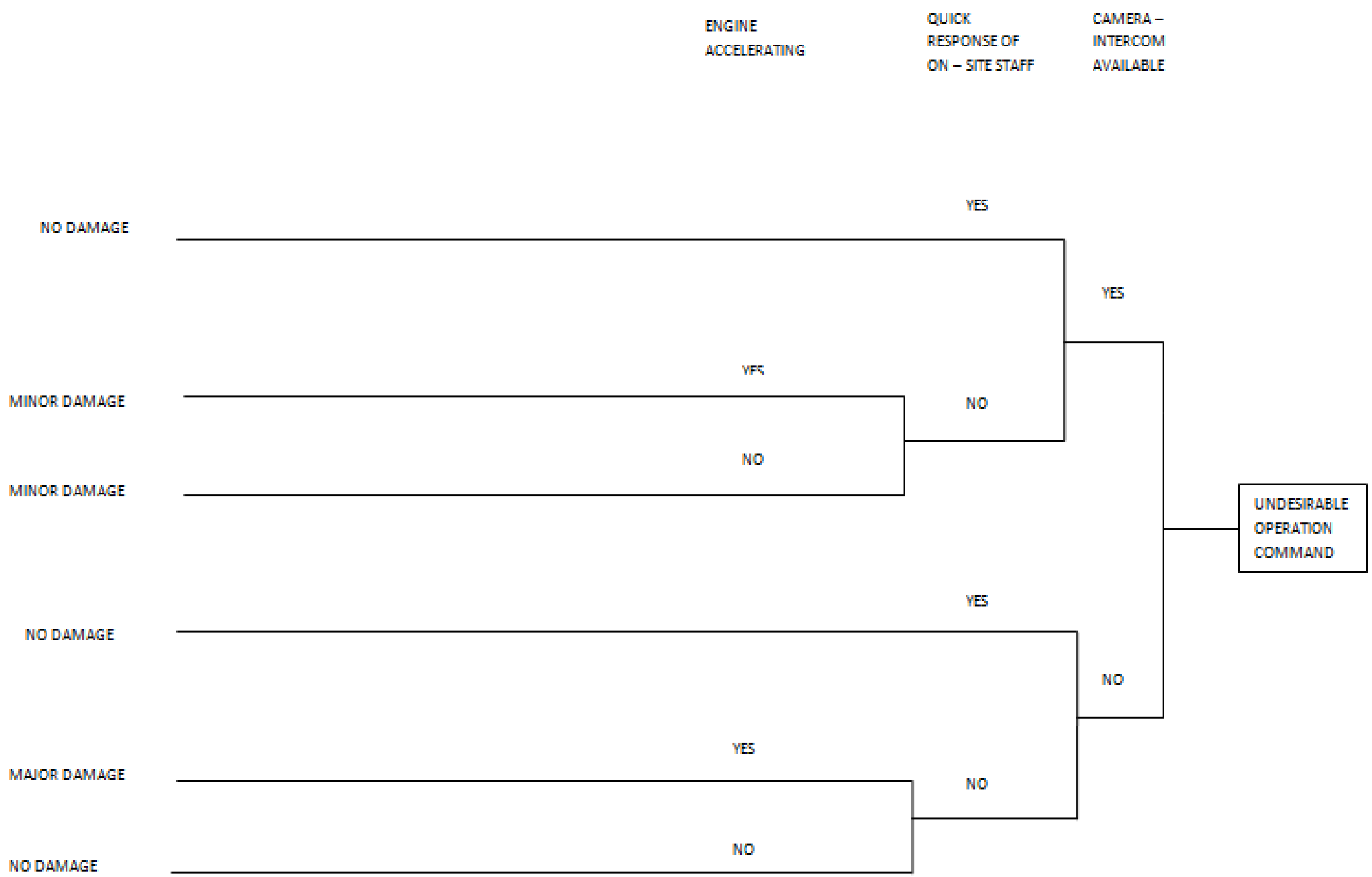


Figure B-4 - ETA for Undesirable Operation Command event

Undesirable Operation Command – Minimal Cut – Sets

P_o: Top undesirable event

P_o = (G+H)[C+(D+C)+(A+B)+(E+F+I+J)]=

= GD+GC+GA+GB+GE+GF+GI+GJ+HC+HD+HA+HB+HE+HF+HI+HJ

Observations:

- Large number of combinations: 16 minimal cut – sets
- Events G and H most dominant – present in several sets (8)
- No triple combinations here – Easier for top event to occur
- Events G and H are also dominant in total control loss event

B.2 HAZOP analysis for the primary design stage of the system

B.2.1 Introduction

HAZOP (Hazard and Operability) analysis in accordance with BS IEC 61882:2001(HAZOP standard) is shown here for physical risk analysis and mitigation of the project in general and for the SRS procedural risk assessment.

Table B-1 - HAZOP analysis of the overall project

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Safeguards	Comments	Actions Required
1.	NO	Switch	Does not interchange circuits	Internal mechanism failure	No manual control available			Simple and reliable mechanism
2.	NO	Control hardware modules system	No signal generation	Unstable Cable connections	Inability to operate through Internet or PC			Secured wire connections
				Device failure				Selection of standardized devices with known failure rate
3.	NO	Measurement acquisition hardware modules system	No signal acquisition	Unstable Cable connections	Wrong indications to the Control Software and User			Secured wire connections
				Device failure				Selection of standardized devices with known failure rate
4.	NO	Control Unit	No interaction with hardware and interface software	Improper connection	Inability to operate through Internet or PC			Secured wire connections
				Device failure				Selection of standardized devices with known failure rate
5.	PART OF	Engine Interface Software	Inadequate communication with CS and hardware	Weak programming	Inconsistent control of the engine through Internet or PC			Thorough testing and validation
				Unstable interaction with CS				Proper establishment of communication between EIS and CS
				Partial communication with control unit				Compatible control unit

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Safeguards	Comments	Actions Required
6.	PART OF	Control Software	Inconsistent interaction with other software components	Triggering of undetected bugs	Wrong response at certain input			Thorough testing and validation
				Malware	System disabled			Malware protection software, Restricted physical access
				DoS	System inoperable			Firewall(s), access through secure VPN, secure transfer protocols
				Exploit attacks	Loss of data, deliberate damage			Firewall(s), access through secure VPN, secure transfer protocols, IDS
				Insider attacks	Loss of data, deliberate damage			Restricted physical access
				User interface not user friendly	Misuse of system from user			Simple and practical UI design, Actual real time data display
7	PART OF	Condition checking system	Partial check of condition parameters	Software disruption as per el. 6	Operation of engine out of limits, improper operation			As per actions 1-5 of el. 6
				Ambient conditions acquisition failure				Reliable hardware and compatible interface software
				Power supply and network connection not checked				Reliable hardware and compatible interface software
8	PART OF	Recorded data storage system	Partial recording of required parameters	As per causes 1-5 el. 6	Loss of recorded data		Not dangerous but important	As per actions 1-5 el. 6
9	NO	Camera system	No image	No power supply	Confusion among users			Power stabiliser
				Mechanical malfunction				Selection of equipment with known failure rate
10	NO	Intercom	No voice communication	No power supply	Confusion among users			Power stabiliser
				Mechanical malfunction				Selection of equipment with known failure rate

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Safeguards	Comments	Actions Required
11	PART OF	Web Service	Poor data transfer	Unsatisfactory network connection	Delay or loss of engine control			Server redundancy, device status detector

Table B-2 - HAZOP analysis for the SRS process

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of communication between involved parties	Insufficient Requirements	Exhaustive meetings and study of the case
2.	NO	Testing of feasibility	Project not checked if it could be constructed	Sufficient information not available	Insufficient Requirements - Often changes of requirements	Research for similar applications
3.	PART OF	System Requirements Clarity	Disambiguity (ambiguity trap)	Generic form of requirements	Often changes of requirements - Time delays	Structured Form of requirements - Validation of requirements
4.	PART OF	Identification of dependencies	Hardware and Software dependencies not addressed	Non functional requirements not clear – Components not directly specified	Insufficient requirements - Time delays	Clarity of System Requirements
5.	NO	Identification of Physical risks	Potential risks of the system during operation not identified	Lack of information – Lack of related applications – Underestimation of the situation	Changes at late stages of design difficult to apply	Derivation of early stage design models of system
6.	PART OF	Cost assessment	Possible Cost not estimated	Requirements not properly defined	Failure to meet requirements	Early stage cost assessment - Evaluation of reusable components (Hardware/Software)

B.3 Round 1 Risk Assessment

B.3.1 Round 1 Physical Risk Assessment

Physical risk assessment of this phase was accomplished by application of Fault Modes and Effect Analysis (FMEA) in conjunction with Fault Modes and Effects Criticality Analysis (FMECA), in accordance with BS EN 60812-2006.

Table B-3 - FMECA for the cDAQ-9174 chassis

Component: cDAQ -9174 chassis								
Failure Mode	Causes	Operating mode	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No interaction with I/O modules	Internal circuit malfunction Embedded Software failure	Operation	No signal generation or readings from the engine	Observe consistency of engine behavior with screen readings after every given command and component identification on the program	Use within specified limits	Marginal (II)	Improbable – score 1 (0 ≤ Pi < 0,001)	Negligible (score 2)
Component inoperative	Power disruption	Operation			Ensure power stability with appropriate devices	Marginal (II)	Occasional – score 3	Negligible (score 3)

Table B-4 – FMECA for the NI 9263 Analogue Output Module

Component: NI 9263 Analog Output module								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No output signal generation	Internal circuit malfunction Embedded Software failure	Operation	No signal generation	Observe consistency of engine behavior with screen readings after every	Use within specified limits	Marginal (II)	Improbable – score 1 (0 ≤ Pi < 0,001)	Negligible (score 2)

Component: NI 9263 Analog Output module								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Component inoperative	Power disruption	Operation		given command	Ensure power stability with appropriate devices	Marginal (II)	Occasional – score 3 (0,01 ≤ Pi < 0,1)	Negligible (score 3)

Table B-5 - FMECA for the serial to USB converter

Component: Tronisoft RS232 to USB cable								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Corrupted data	Internal wiring – connectors chaffing or loosness	Operation	Incorrect flow indication	Compare indication read on the PC with expected values for each position of throttle setting	Use within specified limits Install securely to avoid accidental dislocation	Insignificant (I)	Remote– score 1 0,001 ≤ Pi < 0,01	Negligible (score 2)

Table B-6 – FMECA for the switch harnesses

Component: AlsZone JR and FUTABA switch harnesses								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Switch does not interchange from PC to manual control or vice versa	Switch circuit looseness	Start or interchanging from PC control to manual or vice versa	Inability to interchange from PC control to manual and vice versa if required	Inspect functionality prior to operation	Install securely and handle with proper care only by authorized staff	Critical (III)	Occasional – score 3 0,01 ≤ Pi < 0,1	Undesirable (score9)
Corrupted or no signal	Connections dislocated or damaged			Observe consistency of engine behavior with screen readings after every given command	Install securely to avoid accidental dislocation. Also avoid custom interventions as much as possible	Critical (III)	Occasional – score 3 0,01 ≤ Pi < 0,1	Undesirable (score9)

Table B-7 - FMECA for the extension harnesses

Component: AlsZone JR and FUTABA extension harnesses								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Corrupted or no signal	Connections dislocated or damaged	Operation	Partial or total loss of control from PC	Visual Inspection prior to operation	Install securely to avoid accidental dislocation. Also avoid custom interventions as much as possible	Critical (III)	Occasional – score 3 0,01 ≤ Pi < 0,1	Undesirable (score9)
				Observe consistency of engine behavior with screen readings after every given command		Critical (III)	Occasional – score 3 0,01 ≤ Pi < 0,1	Undesirable (score9)

Table B-8 - FMECA for the connected PC

Component: PC RM EXPERT 3000 **P/N: 181659(SGX)** **S/N: W047392103** **Processor: Intel Pentium 4HT**								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Motherboard failure	Electric power supply peaks	Operation	No connection with the engine and no monitoring	Monitor the voltage of incoming power supply	Ensure power stability with appropriate devices	Marginal (II)	Remote – score 2 (0,001 ≤ Pi < 0,01)	Tolerable (score 4)
Non deliberate shut down	Electric power supply disruption	Operation				Marginal (II)	Occasional – score 3 (0,01 ≤ Pi < 0,1)	Undesirable (score 9)

Table B-9 - FMECA for the EIS

Component: LabVIEW engine interface program*								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Crash after input of boundary values	Undetected bugs – exceeding array boundaries	Operation	Loss of control and monitoring of the engine from the PC	Program not responding and engine behavior does not match screen indications	Verification and Validation of the designed VI. Unit testing, Integration testing of sub vi's, boundary value testing and structural testing	Critical (III)	Not Available	Not Available
Engine monitoring parameters loss	Inadequate integration of relevant VI's for serial data input	Operation	Improper monitoring of the engine with possible erroneous command to follow	Observe consistency of engine behavior with screen readings after every given command		Marginal (II)	Not Available	Not Available
No signal generation from the I/O module	Incomplete set up of VI's and respective ID's	Operation	Loss of control of the engine from the PC			Marginal (II)	Not Available	Not Available

* Frequency of failures and consequently risk scores will be available after extensive testing during beta version of application release

B.3.2 Round 1 Procedural Risk Assessment

Table B-10 – HAZOP process risk assessment for round 1

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
2.	PART OF	Testing and Validation	Inadequate validation	Lack of experience in the use of applied programming language	Integration problems – requirements not properly met	Perform unit testing and Integration tests of the sub components of which EIS comprises
3.	PART OF	Identification of hardware dependencies	Appropriate Hardware not purchased	Objectives not understood or inadequate resource search	Delay of the design	Thorough study of the objectives and the resources of similar applications
4.	NO	Identification of software dependencies	Communication with programs in different language not investigated	No similar applications immediately	Delay of the design – Changes in later stages of the process	Research for similar applications in the literature – practical investigation of interfacing approaches
5.	NO	Identification of Physical risks	Potential risks of the system during operation not identified	Lack of information – Lack of related applications – Underestimation of the situation	Changes at late stages of design difficult to apply	Physical risk assessment and identification of hazards – appropriate mitigation
6.	PART OF	Cost assessment	Implementation cost excessively high	Narrow research of the market and the University for existing and available resources	Restrictive cost which will lead to redesign affecting the whole software process	Investigation for reusable existing components (hardware or software) – Wide search of the market

B.4 Round 2 Risk Assessment

No new hardware was introduced at this round of the Software Process. Therefore, only procedural risk assessment was conducted with the application of HAZOP.

Table B-11 – HAZOP procedural risk assessment for round 2

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
2.	PART OF	Testing and Validation	Inadequate validation	Not enough time	Integration problems – requirements not properly met	Time management - Perform unit testing and Integration tests of the sub components of which EOCS comprises
3.	PART OF	Installation of software components	Components not installed appropriately	Implications of different aspects of installation not understood or inadequate resource search	Performance reduction or security issues	Thorough study of the options and techniques for estimating propagation delay through networks
4.	NO	Identification of software dependencies	Communication with EIS incomplete	No similar applications immediately	Delay of the design – Changes in later stages of the process	Research for similar applications in the literature – practical investigation of interfacing approaches
6.	PART OF	Cost assessment	Implementation cost excessively high	Wide research for existing and available resources	Restrictive cost which will lead to redesign affecting the whole software process	Investigation for reusable existing components (hardware or software) – Wide search of the market

B.5 Round 3 Risk assessment

B.5.1 Round 3 procedural risk assessment

Table B-12 - HAZOP procedural risk assessment for round 3

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
2.	PART OF	Testing and Validation	Inadequate validation	Not enough time	Integration problems – requirements not properly met	Time management - Perform unit testing and Integration tests of the sub components of which EOCS comprises

3.	NO	Identification of Physical risks	Potential risks of the system during operation not identified	Lack of information – Lack of related applications – Underestimation of the situation	Changes at late stages of design difficult to apply	Physical risk assessment and identification of hazards – appropriate mitigation
4.	NO	Identification of software dependencies	Communication with between modules incomplete	No similar applications	Delay of the design – Changes in later stages of the process	Research for similar applications in the literature – practical investigation of interfacing approaches
5.	NO	Identification of hardware requirements	Incomplete functionality of safety features	No similar applications	Delay of the design – Changes in later stages of the process	Research for similar applications in the literature – similar practical examples and manuals of suggested hardware components
6.	PART OF	Cost assessment	Implementation cost excessively high	Narrow research of the market and the University for existing and available resources	Restrictive cost which will lead to redesign affecting the whole software process	Investigation for reusable existing components (hardware or software) – Wide search of the market

B.5.2 Round 3 physical risk assessment

The method applied was Fault Modes and Effect Analysis (FMEA) in combination with Fault Modes and Effects Criticality Analysis (FMECA), in accordance with BS EN 60812-2006.

Table B-13 – FMEA and FMECA for NI 9401 Digital I/O module

Component: NI 9401 Digital I/O module								
Failure Mode	Causes	Operating mode	Failure effects	Failure detection method	Mitigation	Severity	Frequency	Risk
No output signal generation	Internal circuit malfunction Embedded Software failure	Operation	No fuel flow signal reception	Observe consistency of engine behavior with screen readings after every given command	Use within specified limits	Insignificant (I)	Improbable – score 1 (0 ≤ *Pi* < 0,001)	Negligible (score 1)
Component inoperative	Power disruption	Operation			Ensure power stability with appropriate devices	Insignificant (I)	Occasional – score 3 (0,01 ≤ *Pi* < 0,1)	Tolerable (score 3)

Table B-14 - FMEA and FMECA for NI 9265 Thermocouple Input module

Component: NI 9265 Thermocouple Input module								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No output signal generation	Internal circuit malfunction Embedded Software failure	Operation	No signal generation	Observe consistency of engine behavior with screen readings after every given command	Use within specified limits	Marginal (II)	Improbable – score 1 (0 ≤ *Pi* < 0,001)	Negligible (score 2)
Component inoperative	Power disruption	Operation			Ensure power stability with appropriate devices	Marginal (II)	Occasional – score 3 (0,01 ≤ *Pi* < 0,1)	Undesirable (score 6)

Table B-15 - FMEA and FMECA for NI 9215 Analog Input module

Component: NI 9215 Analog Input module								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No output signal generation	Internal circuit malfunction Embedded Software failure	Operation	No signal reception	Inspect before operation	Use within specified limits	Marginal (II)	Improbable – score 1 (0 ≤ *Pi* < 0,001)	Negligible (score 2)
Component inoperative	Power disruption	Operation			Ensure power stability with appropriate devices	Marginal (II)	Occasional – score 3 (0,01 ≤ *Pi* < 0,1)	Undesirable (score 6)

Table B-16 - FMEA and FMECA for NI 9924 25 pin DSUB to terminal connector box

Component: NI 9924 25 pin DSUB to terminal connector box								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No signal reception	Internal circuit malfunction	Operation	No fuel flow signal input	Observe consistency of engine behavior with screen readings after every given command and check condition of connected components via LabVIEW	Use within specified limits	Insignificant (I)	Improbable – score 1 (0 ≤ Pi < 0,001)	Negligible (score 1)
Connection chaffing	Improper wiring by the user	Operation	Improper fuel flow signal input		Follow the manufacturer's instructions properly		Occasional – score 3 (0,01 ≤ Pi < 0,1)	Tolerable (score 3)
Component inoperative	Power disruption	Operation	No fuel flow signal input		Ensure power stability with appropriate devices		Occasional – score 3 (0,01 ≤ Pi < 0,1)	Tolerable (score 3)

Table B-17 - FMEA and FMECA for Floscan 201-A6 Fuel Flow Transducer

Component: Floscan 201-A6 Fuel Flow Transducer								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
No signal	Internal rotor failure	Operation	No fuel flow indication	Compare indication read on the PC with expected values for each position of throttle setting	Use within specified limits	Insignificant (I)	Remote– score 1 0,001 ≤ Pi < 0,01	Negligible (score 2)
Incorrect flow indication	Improper connection by the user or wrong adjustment of the K-Factor	Operation	Incorrect fuel flow indication		Follow the manufacturer's instructions properly		Occasional – score 3 (0,01 ≤ Pi < 0,1)	Tolerable (score 3)

Table B-18 - FMEA and FMECA for Trust EXIS Chat Pack webcam

Component: Trust EXIS Chat Pack webcam								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Failure to start	Not triggered by used software (OpenCV)	Operation	No live video image	Run the program	Restart program	Insignificant (I)	Occasional – score 3 (0,01 ≤ *Pi* < 0,1)	Tolerable (score 3)
Mechanical failure	Vibration and noise in the test house	Operation	No live video image	Check daily before first run of the engine	Place in safe distance from the engine	Insignificant (I)	Remote– score 1 0,001 ≤ *Pi* < 0,01	Negligible (score 2)

Table B-19 - FMEA and FMECA for Maplin 500mA AC to AC Mains Adaptor, N57AT

Component: Maplin 500mA AC to AC Mains Adaptor, N57AT								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Internal failure	Power surge	Start or Operating	Impossible to determine voltage of main supply	Inspect functionality prior to operation	Redundancy	Critical (III)	Improbable – score 1 0,001 ≤ *Pi* < 0,01	Tolerable (score 3)

Component: Maplin 500mA AC to AC Mains Adaptor, N57AT								
Failure Mode	**Causes**	**Operating mode**	**Failure effects**	**Failure detection method**	**Mitigation**	**Severity**	**Frequency**	**Risk**
Connection failure	Connections not properly secured		Unstable or no indication of voltage of main supply	Inspect functionality prior to operation	Install securely to avoid accidental dislocation. Also avoid custom interventions as much as possible	Critical (III)	Occasional – score 3 0,01 ≤ *Pi* < 0,1	Undesirable (score9)

Table B-20 – FMEA and FMECA for Maplin LED indication lights 60.13.C1

Component: Maplin LED indication lights 60.13.C1								
Failure Mode	Causes	Operating mode	Failure effects	Failure detection method	Mitigation	Severity	Frequency	Risk
Internal failure	Power surge	Start or Operating	Impossible to indicate need for manual control reception	Inspect functionality prior to operation	Redundancy	Critical (III)	Remote– score 1 0,001 ≤ *Pi* < 0,01	Tolerable (score 3)

Component: Maplin LED indication lights 60.13.C1								
Failure Mode	Causes	Operating mode	Failure effects	Failure detection method	Mitigation	Severity	Frequency	Risk
Connection failure	Connections not properly secured		Unstable or no indication of need for manual control reception	Inspect functionality prior to operation	Install securely to avoid accidental dislocation. Also avoid custom interventions as much as possible	Critical (III)	Occasional – score 3 $0{,}01 \leq Pi < 0{,}1$	Undesirable (score9)

B.6 Round 4 Risk Assessment

Only procedural risk assessment was applicable at this round.

Table B-21 - HAZOP procedural risk assessment for round 3

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
2.	PART OF	Testing and Validation	Inadequate validation	Not enough time	Integration problems – requirements not properly met	Time management - Perform unit testing and Integration tests of the sub components and the integrated web application
3.	NO	Identification of Internet connection risks	Bad connections or disrupted networks not considered	Slow networks – Abrupt power loss – intermediate server failure	Unreliable final application	Evaluate the problem and accommodate safety features at appropriate positions

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
4.	NO	Identification of software dependencies	Communication with application from browser incomplete	No similar applications	Delay of the design – Modifications	Research for similar applications in the literature – practical investigation of web applications and web services
5.	PART OF	Cost assessment	Implementation cost excessively high	Narrow research of the market and the University for existing and available resources	Restrictive cost which will lead to redesign affecting the whole software process	Investigation for reusable existing components (hardware or software) – Wide search of the market

B.7 Round 5 Risk Assessment

Only procedural risk assessment was applicable at this round.

Table B-22 - HAZOP procedural risk assessment for round 5

No	Guide Word	Element	Deviation	Possible Causes	Consequences	Actions Required
1.	PART OF	Objectives Identification	Not clearly stated and realised	Lack of understanding of the requirements	Partial fulfillment of the requirements	Thorough study of the requirements an clear breakdown of the objectives
2.	PART OF	Selection of validation method	Not easily applicable	Lack of experience in formal validation methods	Delay and insufficient validation – Problems at released version of application	Extensive research about available methods and software tools in safety critical systems and the aviation industry
3.	PART OF	Derivation of test cases	Insufficient test cases to obtain 100% DC	Improper static analysis of the cod – partial identification of application states and transitions	Insufficient validation – Problems at released version of application	Thorough static analysis of the code - Revisit the validation of the individual modules – Study the requirements accomplishment
4.	PART OF	Validation coverage	Insufficient coverage	Ambiguities of the requirements – Inadequate test cases – partial identifications of application states	Insufficient validation – Problems at released version of application	Revisit and finalization of the requirements - Revisit the validation of the individual modules – Thorough state models creation – Perform additional final testing beyond the requirements testing to ensure 100% DC coverage
5.	PART OF	Cost assessment	Use of non open source software tools for validation - cost excessively high	Lack of experience in formal validation methods - unsuitable selection of method due to lack of experience	Restrictive cost which will lead to delay of release and problems after	Research for similar application in the aviation industry – Definition of suitable approach - Investigation for open source tools

Appendix C Software Components Control Flow Validation

This Appendix is divided in 3 main sections: LabVIEW components, Java SE components and the JavaScript client. The control flow diagram for each module is shown and based on these the cyclomatic complexity was calculated and used as guidance for the identification of the possible flow paths shown in the corresponding validation tables. The latter include the test derived test cases with the required assertions.

C.1 LabVIEW components

C.1.1 Switch Signal function

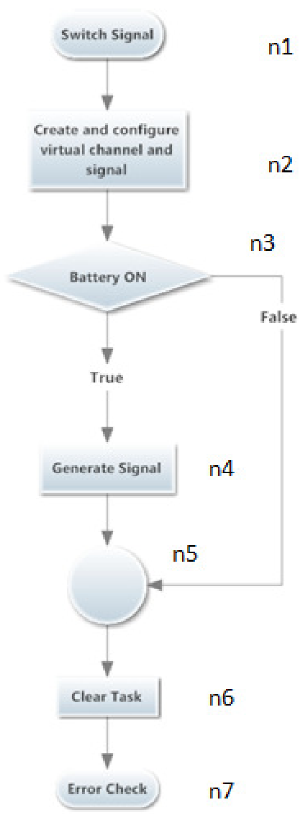


Figure C-1 – EIS.VI Switch Signal function control flow

Nodes: n = 7
Edges: e = 7
Cyclomatic Complexity:
VG = e – n + 2 = 7 – 7 + 2 = 2
Individual flow paths: 2

Table C-1 – Test cases for EIS.VI Switch Signal function

Test Case	Path	Description	Assertions
1	{n1n2n5n6n7}	Switch selected, Battery = False	No signal generated
2	{n1n2n3n4n5n6n7}	Switch selected, Battery = True	Signal generated

Note: Output signal measured and found to be according to manufacturer's specification for all 3 positions of the switch knob.

C.1.2 Throttle Signal function

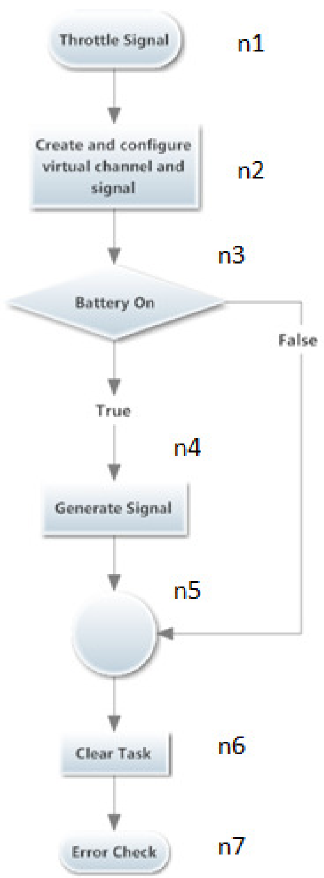


Figure C-2 - EIS.VI Throttle Signal function control flow

Nodes: n = 7
Edges: e = 7
Cyclomatic Complexity:
VG = e – n + 2 = 7 – 7 + 2 = 2
Individual flow paths: 2

Table C2 - Test cases for EIS.VI Throttle Signal function

Test Case	Path	Description	Assertions
1	{n1n2n5n6n7}	Switch selected, Battery = False	No signal generated
2	{n1n2n3n4n5n6n7}	Switch selected, Battery = True	Signal generated

Note: As throttle slider can take 101 positions, boundary values of output signal were measured and found to be according to manufacturer's specification. Also selective values of output signal within regular intervals of 20% of the throttle movement were measured and found to be according to manufacturer's specification.

C.1.3 EIS.vi ECU Serial function

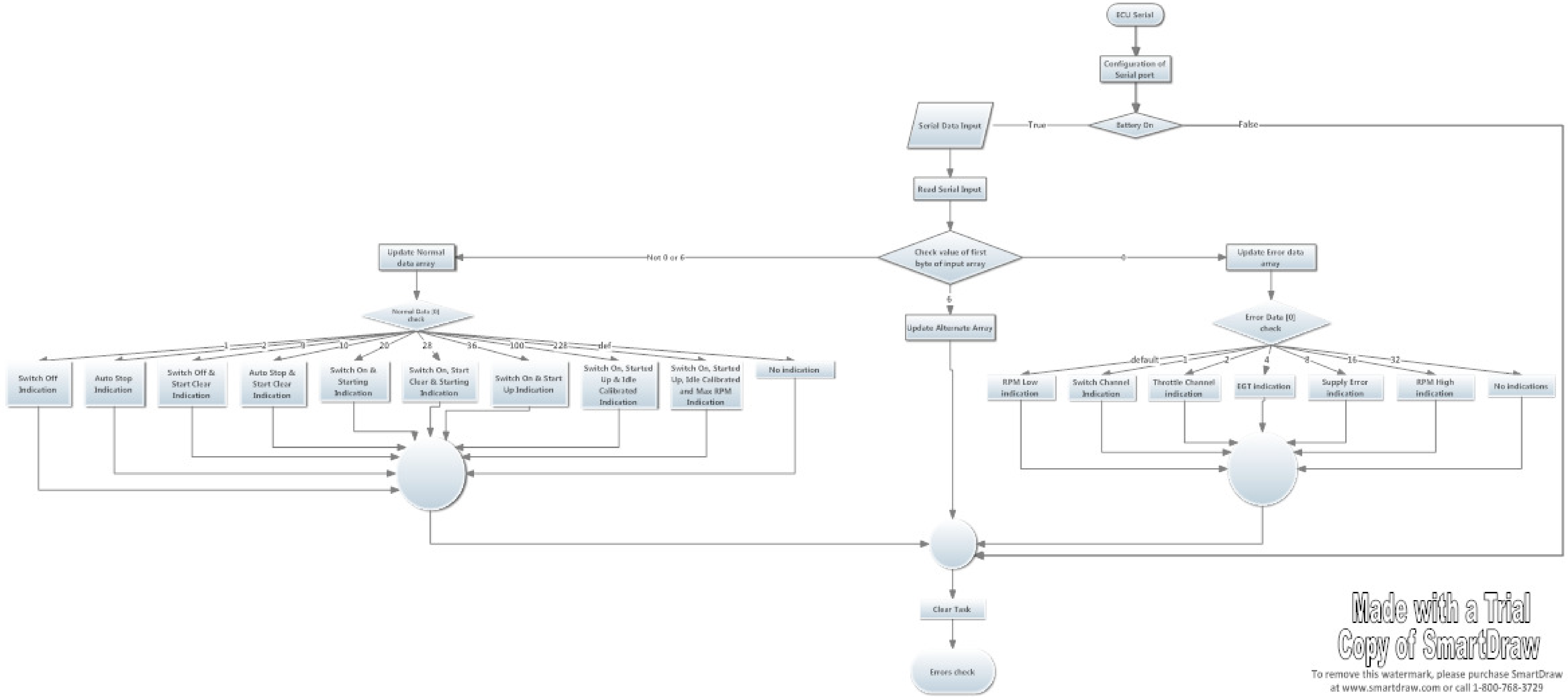


Figure C-3 - EIS.vi ECU Serial function control flow

Nodes: n = 33
Edges: e = 50
Cyclomatic Complexity:
VG = e – n + 2 = 50 – 33 + 2 = 19
Individual flow paths: 19

Table C-3 - Test cases for EIS.vi ECU Serial function

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n5n6n9n11n12n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=1	EGT, RPM, throttle, Vout updated - switch off indication
2.	{n1n2n3n4n5n6n9n11n13n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=2	EGT, RPM, throttle, Vout updated - auto stop indication
3.	{n1n2n3n4n5n6n9n11n14n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=9	EGT, RPM, throttle, Vout updated - switch off & start clear indication
4.	{n1n2n3n4n5n6n9n11n15n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=10	EGT, RPM, throttle, Vout updated - auto start & start clear indication
5.	{n1n2n3n4n5n6n9n11n16n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=20	EGT, RPM, throttle, Vout updated - switch on & starting indication
6.	{n1n2n3n4n5n6n9n11n17n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=28	EGT, RPM, throttle, Vout updated - switch on & start clear & starting indication
7.	{n1n2n3n4n5n6n9n11n18n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=36	EGT, RPM, throttle, Vout updated - switch on & started up indication
8.	{n1n2n3n4n5n6n9n11n19n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=100	EGT, RPM, throttle, Vout updated - switch on & started up & idle calibrated indication
9.	{n1n2n3n4n5n6n9n11n20n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]=228	EGT, RPM, throttle, Vout updated - switch on & started up & idle calibrated & Max RPM indication
10.	{n1n2n3n4n5n6n9n11n11n22n8n32n33}	Battery = True, array[0]!=0|6, normalData[0]= any other value	EGT, RPM, throttle, Vout updated - no engine status indication
11.	{n1n2n3n4n5n6n7n8n32n33}	Battery = True, array[0]=6	Vsupply updated
12.	{n1n2n3n8n32n33}	Battery = False	No indications and updates
13.	{n1n2n3n4n5n6n10n23n24n31n32n33}	Battery = True, array[0]=0,errorData[0]=1	EGT, RPM, throttle, Vout updated - RPM low indication
14.	{n1n2n3n4n5n6n10n23n25n31n32n33}	Battery = True, array[0]=0,errorData[0]=2	EGT, RPM, throttle, Vout updated - Switch Channel not

Test Case	Path	Description	Assertions
			present indication
15.	{n1n2n3n4n5n6n10n23n26n31n32n33}	Battery = True, array[0]=0,errorData[0]=4	EGT, RPM, throttle, Vout updated - Throttle Channel not present indication
16.	{n1n2n3n4n5n6n10n23n27n31n32n33}	Battery = True, array[0]=0,errorData[0]=8	EGT, RPM, throttle, Vout updated - EGT High indication
17.	{n1n2n3n4n5n6n10n23n28n31n32n33}	Battery = True, array[0]=0,errorData[0]=16	EGT, RPM, throttle, Vout updated - Supply Voltage indication
18.	{n1n2n3n4n5n6n10n23n29n31n32n33}	Battery = True, array[0]=0,errorData[0]=32	EGT, RPM, throttle, Vout updated - RPM High indication
19.	{n1n2n3n4n5n6n10n23n30n31n32n33}	Battery = True, array[0]=0,errorData[0]=any other value	EGT, RPM, throttle, Vout updated - No error indication

C.1.4 EIS.vi integration testing

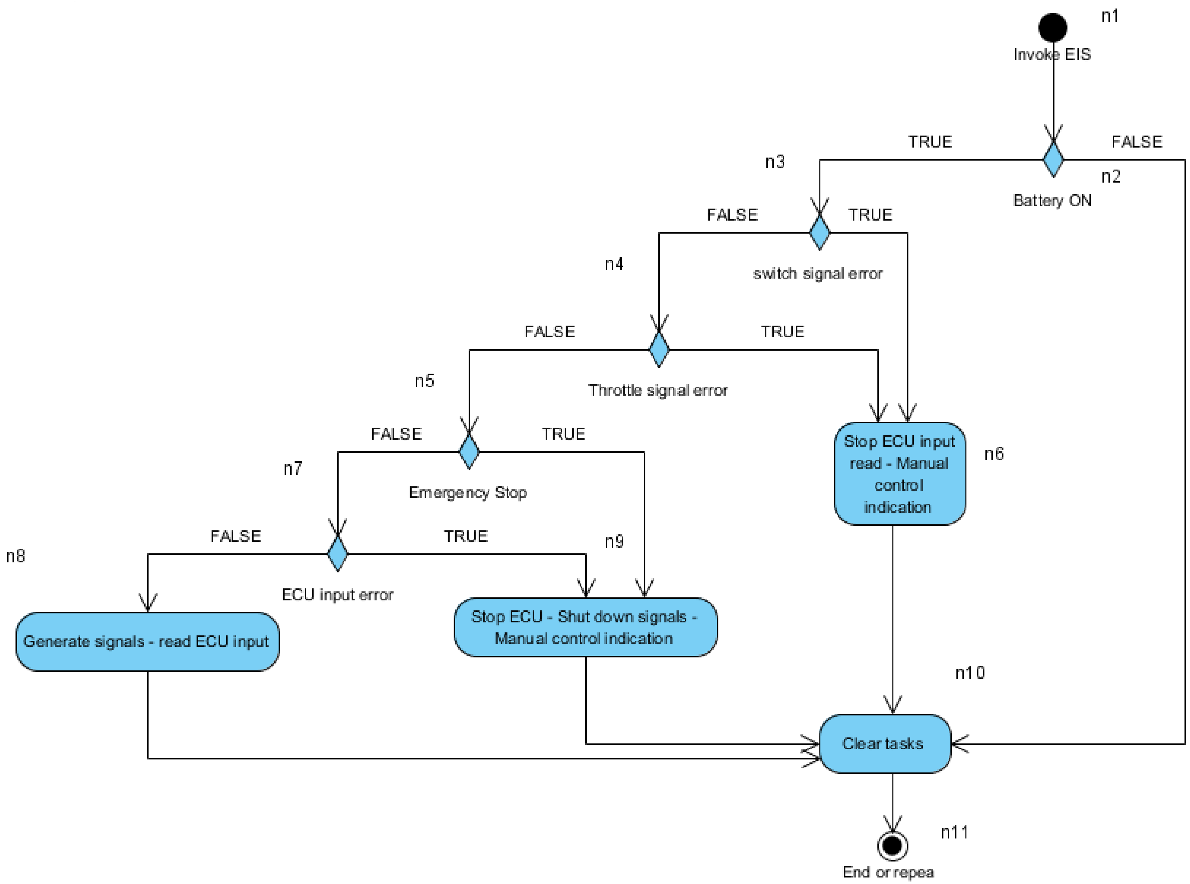


Figure C-4 – EIS.vi Integration testing control flow diagram

Nodes: n = 11
Edges: e = 15
Cyclomatic Complexity:
VG = e – n + 2 = 15 – 11 + 2 = 6
Individual flow paths: 6

Table C-4 - Test cases for EIS.vi integration testing

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n17n18n19n20}	Start RMI server – start MEOCS – close MEOCS	MEOCS closed – RMI server reset and ready
2.	{n1n2n3n4n5n7n8n12 n17n18n19n20}	RMI Server running – start MEOCS – send battery, throttle or switch command	General Test Harness settings change accordingly
3.	{ n1n2n3n4n5n7n9n12n17n18n19n20}	RMI Server running – start MEOCS – send corrupted command using TCPClient harness	General Test Harness settings remain unchanged
4.	{n1n2n3n4n5n6n10n13n14n15n17n18n19n20}	RMI Server running – start MEOCS – Alter the output message of General Test Harness in a wrong format – Send any command from MEOCS	MEOCS indications do not change
5.	{ n1n2n3n4n5n6n10n13n14n15n16n17n18n19n20}	RMI Server running – start MEOCS – Alter the output message of General Test Harness in a wrong format – Alter the timeout of reception socket in class MessageImplementation - Send any command from MEOCS	MEOCS and EISI shut down - RMI server reset and ready
6.	{n1n2n3n4n5n6n10n11n14n15n17n18n19n20}	RMI Server running – start MEOCS - Send any command from MEOCS	MEOCS and EISI shut down - RMI server reset and ready
7.	{n1n2n3n4n5n6n10n11n14n15n16n17n18n19n20}	RMI Server running – start MEOCS - Send any command from MEOCS	General Test Harness settings change accordingly

C.1.5 EIS System Status Determination function

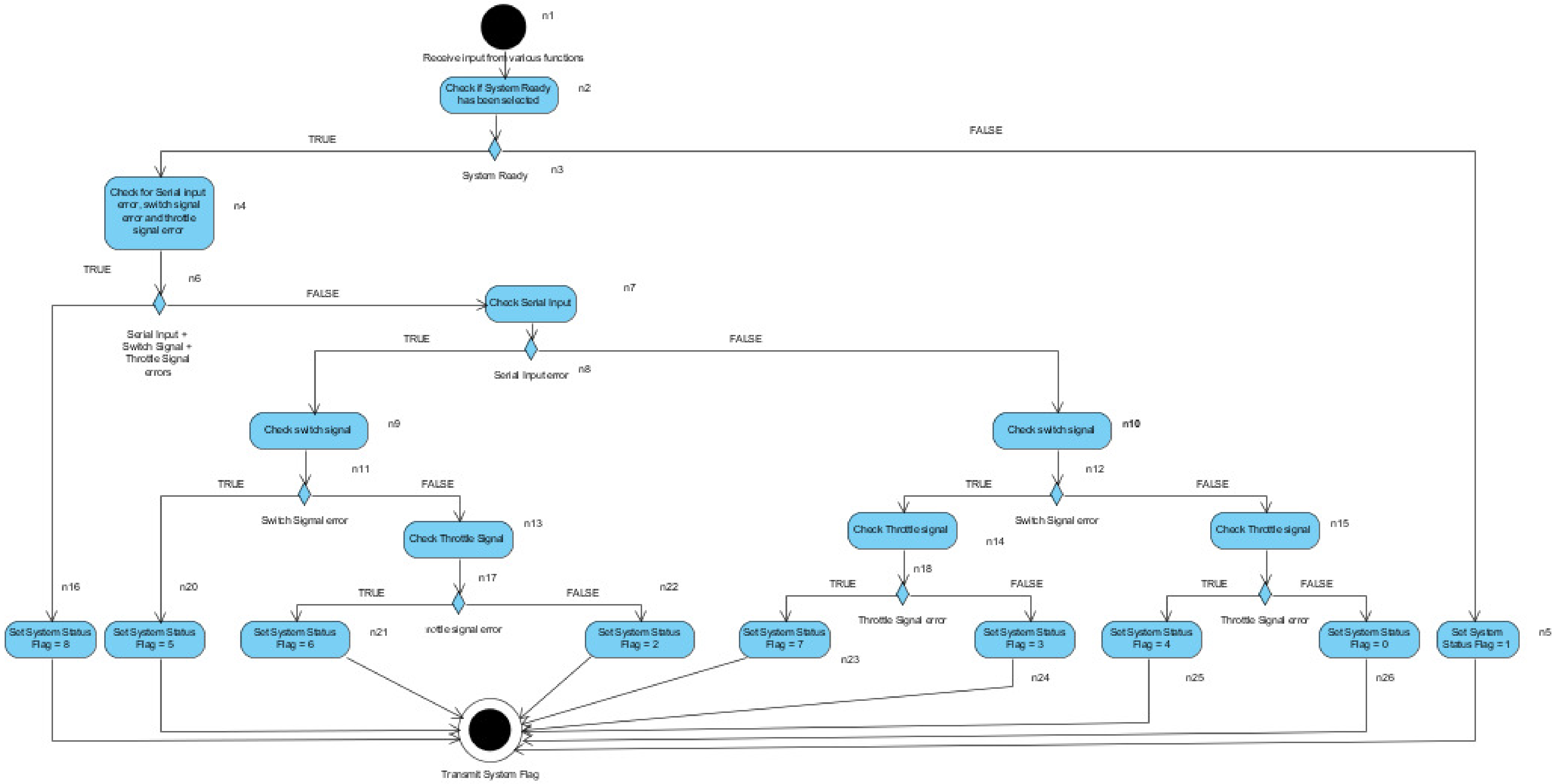


Figure C-5 – System Status Determination function control flow

Nodes: n = 34
Edges: e = 27
Cyclomatic Complexity:
VG = e – n + 2 = 34 – 27 + 2 = 9
Individual flow paths: 9

Table C-5 - Test cases for System Status Determination function

Test Case	Path	Description	Assertions
1.	{n1n2n3n5n27}	systemReady = FALSE	systemStatusFlag = 1
2.	{n1n2n3n4n6n16n27}	systemReady & Node6 = TRUE	systemStatusFlag = 8
3.	{n1n2n3n4n6n7n8n9n11n20n27}	systemReady = TRUE & Node6* = FALSE & serialInputError = TRUE & switchSignalError = TRUE	systemStatusFlag = 5
4.	{ n1n2n3n4n6n7n8n9n11n13n17n21n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = TRUE & switchSignalError = FALSE & throttleSignalError = TRUE	systemStatusFlag = 6
5.	{n1n2n3n4n6n7n8n9n11n13n17n22n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = TRUE & switchSignalError = FALSE & throttleSignalError = FALSE	systemStatusFlag = 2
6.	{n1n2n3n4n6n7n8n9n12n14n18n23n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = FALSE & switchSignalError = TRUE & throttleSignalError = TRUE	systemStatusFlag = 7
7.	{ n1n2n3n4n6n7n8n9n12n14n18n24n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = FALSE & switchSignalError = TRUE & throttleSignalError = FALSE	systemStatusFlag = 3
8.	{n1n2n3n4n6n7n8n9n12n15n19n25n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = FALSE & switchSignalError =FALSE & throttleSignalError = TRUE	systemStatusFlag = 4
9.	{ n1n2n3n4n6n7n8n9n12n15n19n26n27}	systemReady = TRUE & Node6 = FALSE & serialInputError = FALSE & switchSignalError =FALSE & throttleSignalError = FALSE	systemStatusFlag = 0

*Node6 = serialInputError & switchSignalError & throttleSignalError

C.1.6 Mains Supply Interface System (MSIS) of AmbientSensing.vi

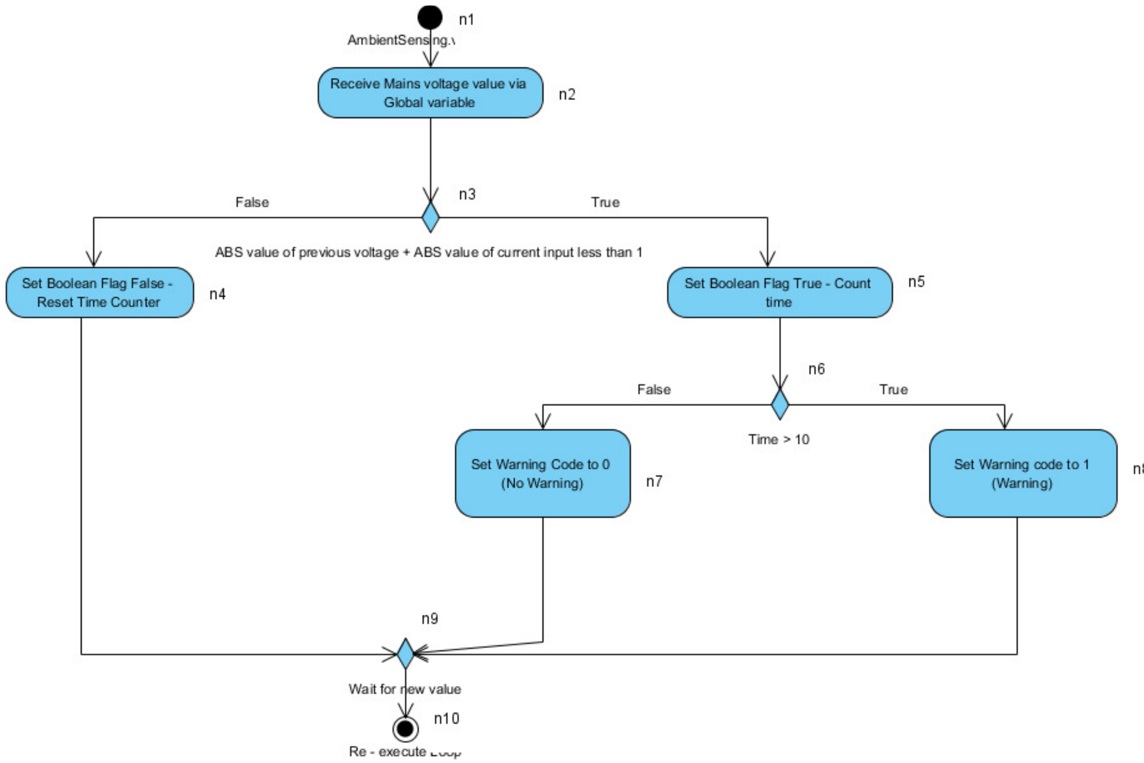


Figure C-6 – Mains Supply Interface System (MSIS) control flow

Nodes: n = 10
Edges: e = 11
Cyclomatic Complexity:
VG = e – n + 2 = 11 – 10 + 2 = 3
Individual flow paths: 3

Table C-6 – Test cases for Mains Supply Interface System (MSIS)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n9n10}	Run function normally	elapsedTimeS = 0 – mainsFlag = 0
2.	{n1n2n3n5n6n7n9n10}	Unplug AC/AC adapter & in time < 10 sec re plug it	elapsedTimeS != 0 – mainsFlag = 0 After re-plugging: elapsedTimeS != 0 – mainsFlag = 0
3.	{ n1n2n3n5n6n8n9n10}	Unplug AC/AC adapter and leave out fot time > 10 sec	elapsedTimeS != 0 – mainsFlag = 0 After 10 sec: elapsedTimeS >10 sec – mainsFlag = 1

C.2 Java SE components

C.2.1 EIS Interface

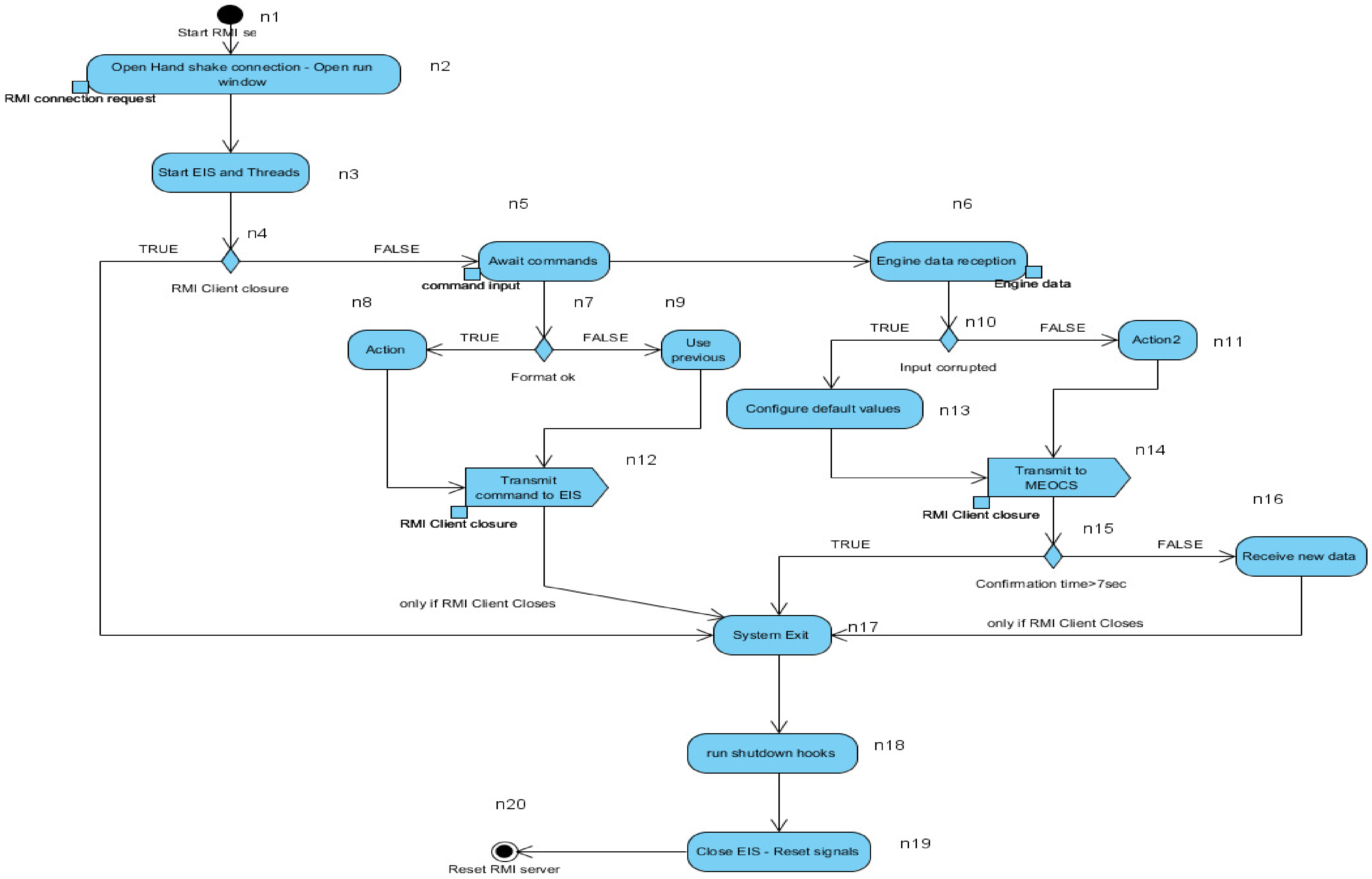


Figure C-7 – EIS Interface (EISI) control flow

Nodes: n = 20
Edges: e = 25
Cyclomatic Complexity:
VG = e – n + 2 = 25 – 20 + 2 = 7
Individual flow paths: 3

Table C-7 – Test cases for EIS Interface (EISI)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n17n18n19n20}	Start RMI server – start MEOCS – close MEOCS	MEOCS closed – RMI server reset and ready
2.	{n1n2n3n4n5n7n8n12 n17n18n19n20}	RMI Server running – start MEOCS – send battery, throttle or switch command	General Test Harness settings change accordingly
3.	{ n1n2n3n4n5n7n9n12n17n18n19n20}	RMI Server running – start MEOCS – send corrupted command using TCPClient harness	General Test Harness settings remain unchanged
4.	{n1n2n3n4n5n6n10n13n14n15n17n18n19n20}	RMI Server running – start MEOCS – Alter the output message of General Test Harness in a wrong format – Send any command from MEOCS	MEOCS indications do not change
5.	{ n1n2n3n4n5n6n10n13n14n15n16n17n18n19n20}	RMI Server running – start MEOCS – Alter the output message of General Test Harness in a wrong format – Alter the timeout of reception socket in class MessageImplementation - Send any command from MEOCS	MEOCS and EISI shut down - RMI server reset and ready
6.	{n1n2n3n4n5n6n10n11n14n15n17n18n19n20}	RMI Server running – start MEOCS - Send any command from MEOCS	MEOCS and EISI shut down - RMI server reset and ready
7.	{n1n2n3n4n5n6n10n11n14n15n16n17n18n19n20}	RMI Server running – start MEOCS - Send any command from MEOCS	General Test Harness settings change accordingly

C.2.2 Main Engine Operation Control Software (MEOCS)

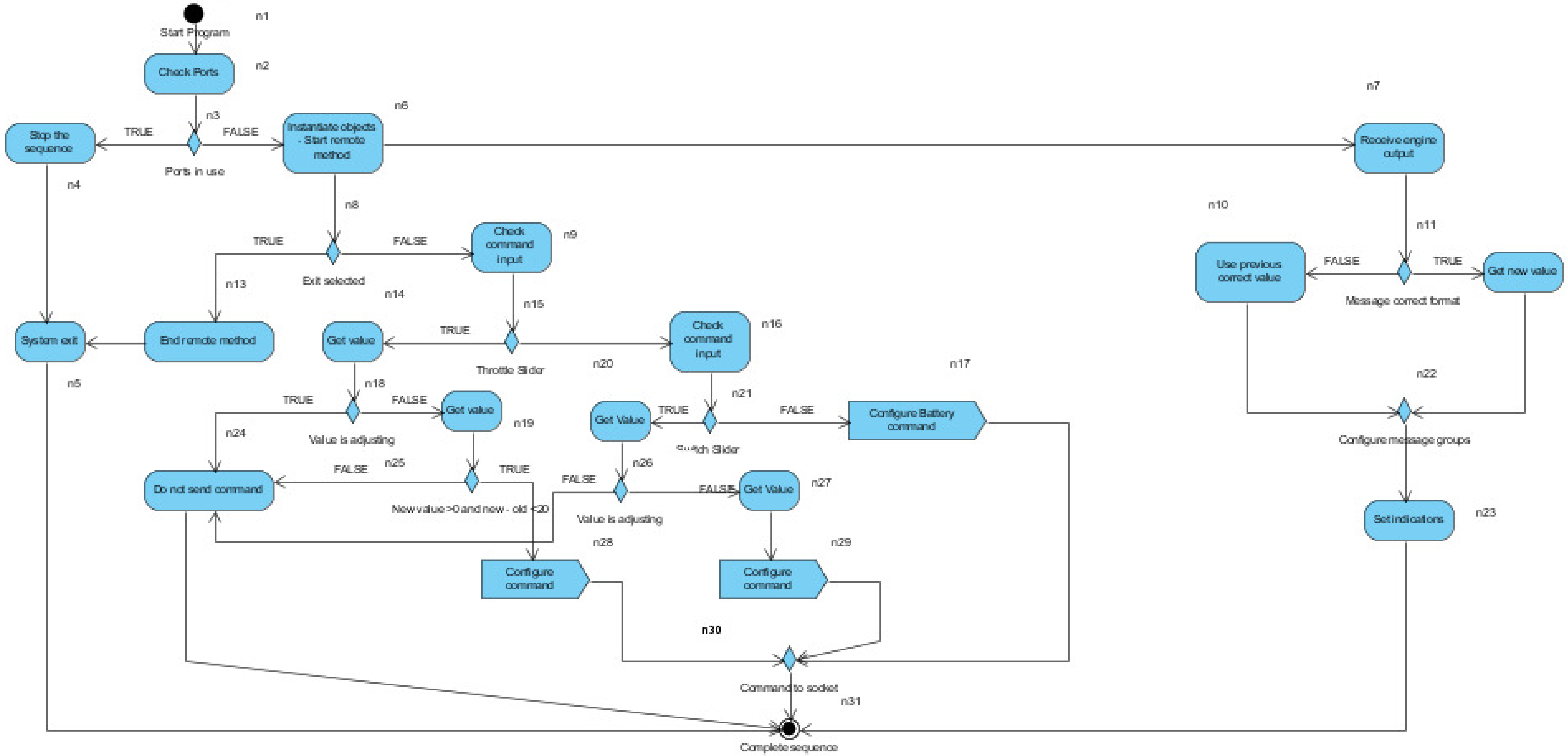


Figure C-8 – Main Engine Operation Control Software (MEOCS) control flow

Nodes: n = 31
Edges: e = 39
Cyclomatic Complexity:
VG = e – n + 2 = 39 – 31 + 2 = 10
Individual flow paths: 10

Table C-8 – Test cases for Main Engine Operation Control Software (MEOCS)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n5n31}	MEOCS already in operation or ports 1111, 4444, 1212, 1099, 3311, 9999 occupied	MEOCS closed
2.	{n1n2n3n6n7n11n10n22n23n31}	Input engine data not as specified format from TCPClient test harness	Indications not change
3.	{ n1n2n3n6n7n11n12n22n23n31}	Input engine data as specified format from TCPClient test harness	Indications change
4.	{ n1n2n3n6n8n13n5n31}	MEOCS running and select exit	MEOCS closed and EISI closed as well
5.	{n1n2n3n6n8n9n15 n14n18n24n31}	Click and hold the throttle slider and move without releasing	Throttle setting of General Test Harness does not change
6.	{n1n2n3n6n8n9n15 n14n18n19n25n24n31}	Insert a throttle increase more than 20% in the throttle slider	Throttle returns to previous position - Throttle setting of General Test Harness does not change
7.	{ n1n2n3n6n8n9n15 n14n18n19n25n28n30n31}	Insert a throttle increase less than 20% in the throttle slider	Throttle setting changes in General Test Harness along with indications
8.	{ n1n2n3n6n8n9n15n16n21n20n26n24n30n31}	Click and hold the switch slider and move without releasing	Switch position indication of General Test Harness does not change
9.	{n1n2n3n6n8n9n15n16n21n20n26n27n29n30n31}	Set the switch slider to a new position	Switch position indication of General Test Harness changes accordingly
10.	{ n1n2n3n6n8n9n15n16n21n17n30n31}	Select the Battery toggle button	Battery position of General Test Harness changes accordingly

C.2.3 Trouble Shooting System (TSS)

Due to the complexity of this module, the control flow testing was conducted in 3 stages: from node 1 to node 26 (CF1), from node 26 to node 39 (CF2) and node 26 analysed into its contained simple actions. Node 26 represents the functionality that conducts the checks regarding the mains power supply status.

Prerequisites for CF1:

1. TCPClient test harness used to send MEOCS format commands
2. TCPServer test harness engaged to accept output from TSS
3. Class TSSCentral in package TSS.tss modified to include main function
4. General Test Harness initiated by TSSCentral main function

Prerequisites for CF2 and Mains Power Supply functionality:

1. Class TSSCentral in package TSS.tss TSSCentral configured with mainTss static method instead of main() function
2. MEOCS, EISI and GTH engaged

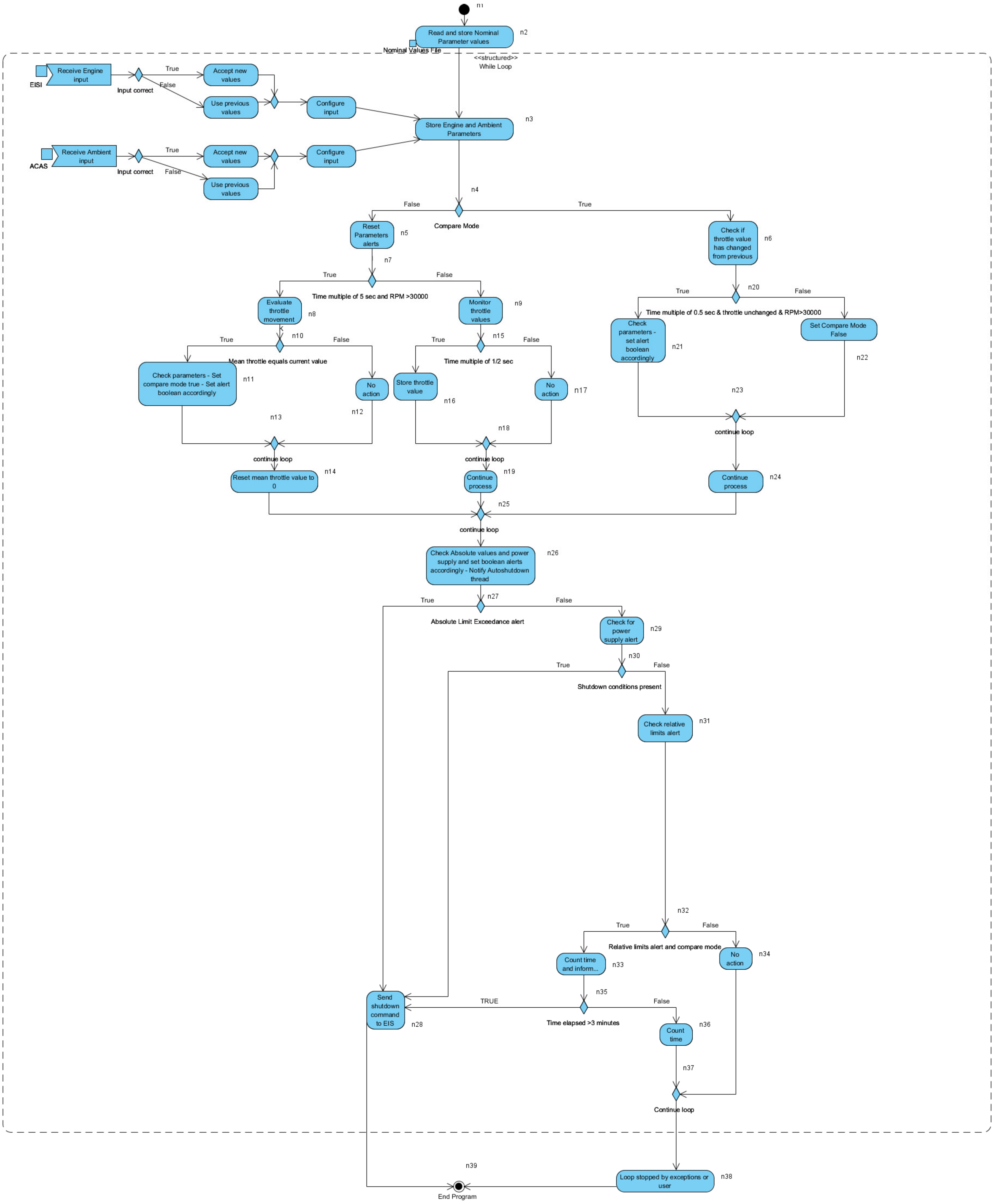


Figure C-9 – Trouble Shooting System (TSS) control flow

CF1 (nodes 1 – 26):

Nodes: n = 26
Edges: e = 30
Cyclomatic Complexity:
VG = e – n + 2 = 30 – 26 + 2 = 6
Individual flow paths: 6

CF2 (nodes 26 - 39):

Nodes: n = 14
Edges: e = 17
Cyclomatic Complexity:
VG = e – n + 2 = 17 – 14 + 2 = 5
Individual flow paths: 5

Table C-9 – Test cases for CF1 of Trouble Shooting System (TSS)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n5n7n8n10n11n13n14n25n26}	GTH simulating engine running - Let throttle setting more than 10 sec stable	elapsed Time = multiple of 10 sec at check point 1 on IDE output field - compareMode = TRUE and throttleReference = 0 at check point 2 on IDE output field
2.	{n1n2n3n4n5n7n8n10n12n13n14n25n26}	GTH simulating engine running - Give throttle change command throttle on TCPClient and bring back to previous position immediately	elapsed Time = multiple of 10 sec at check point 1 on IDE output field - compareMode = FALSE and throttleReference = 0 at check point 2 on IDE output field
3.	{n1n2n3n4n5n7n9n15n16n18n19n25n26}	GTH simulating engine running - Let throttle setting more than 10 sec stable	elapsed Time = !multiple of 10 sec at check point 1 on IDE output field – elapsedTime2 = multiple of 0.5 sec at check point 3 on IDE output - throttleReference = value.equals throttle setting – check point 4 on IDE output indicates "throttleReference Updated"
4.	{n1n2n3n4n5n7n9n15n17n18n19n25n26}	GTH simulating engine running - Let throttle setting more than 10 sec stable	elapsed Time = !multiple of 10 sec at check point 1 on IDE output field – elapsedTime2! = multiple of 0.5 sec at check point 3 on IDE output - throttleReference = value.equals throttle setting – check point 4 on IDE output indicates "throttleReference not Updated"
5.	{n1n2n3n4n6n20n21n22n23n24n25n26}	GTH simulating engine running - Let throttle setting more than 10 sec stable	compareMode = TRUE at check point 2 on IDE output field - check point 5 on IDE output indicates "Booleans set"
6.	{n1n2n3n4n6n20n22n23n24n25n26}	GTH simulating engine running - Let throttle on MEOCS more than 10 sec stable – Give throttle change command throttle on TCPClient	compareMode = FALSE at check point 6 on IDE output

Table C-10 - Test cases for CF1 of Trouble Shooting System (TSS)

Test Case	Path	Description	Assertions
1.	{n26n27n28n29}	GTH simulating engine running - On GTH panel insert RPM or EGT ≥ absolute limit	Respective red warning illuminates on MEOCS panel and system shutdown – RMI server ready to accept connection
2.	{ n26n27n29n30n28n39}	GTH simulating engine running - On GTH panel insert MAINS OFF and no commands for 3 min	Respective yellow warning illuminates on MEOCS panel – After 3 min elapsed time system shutdown – RMI server ready to accept connection
3.	{n26n27n29n30n31n32n33n35n28n39}	GTH simulating engine running - On GTH panel insert RPM or EGT > relative limit for current throttle setting and no commands for 3 min	Respective yellow warning illuminates on MEOCS panel – After 3 min elapsed time system shutdown – RMI server ready to accept connection
4.	{ n26n27n29n30n31n32n33n35n36n37n38n39}	GTH simulating engine running - On GTH panel insert RPM or EGT > relative limit for current throttle setting and no commands for 2.5 min	Respective yellow warning illuminates on MEOCS panel
5.	{ n26n27n29n30n31n32n34n37n38n39}	GTH simulating engine running – hit exit on MEOCS panel	system shutdown – RMI server ready to accept connection

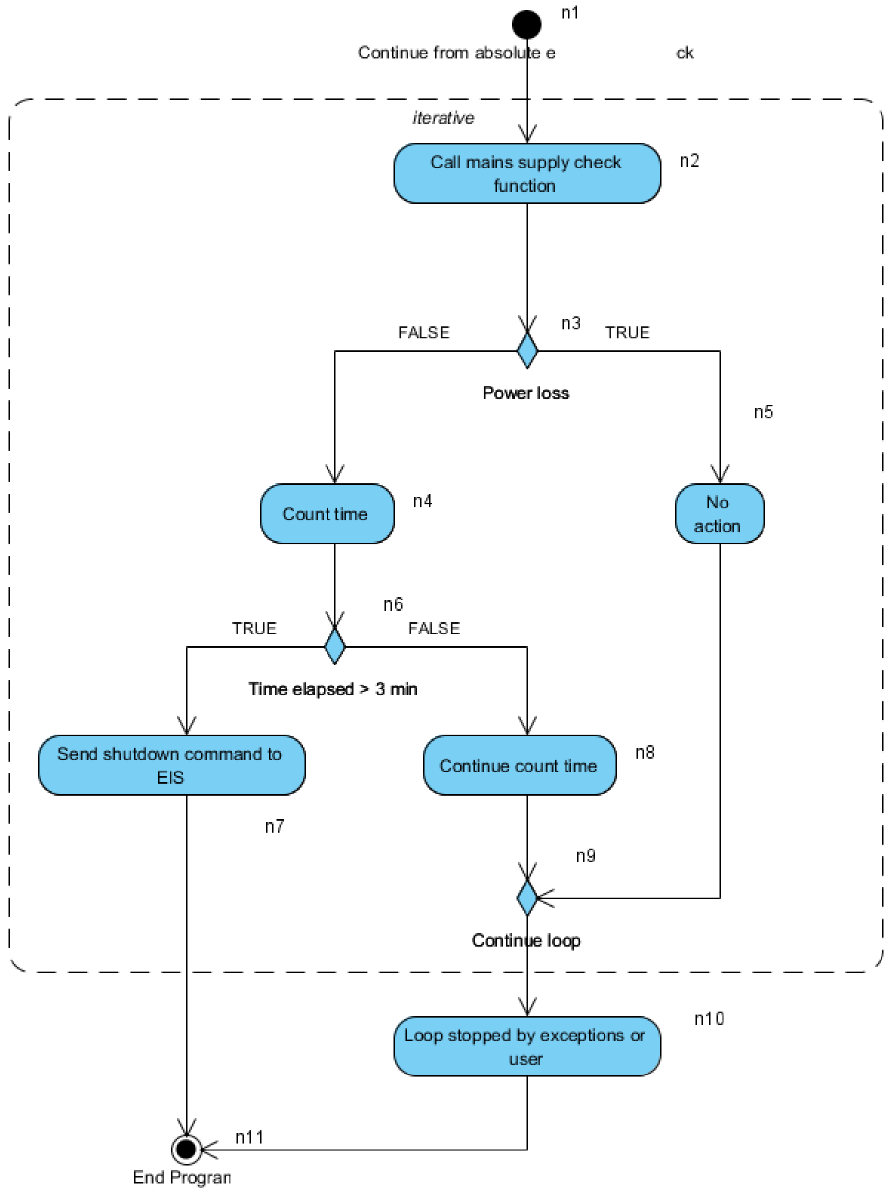


Figure C-10 – Mains Power Supply functionality of TSS - control flow

Nodes: n = 11
Edges: e = 12
Cyclomatic Complexity:
VG = e – n + 2 = 12 – 11 + 2 = 3
Individual flow paths: 3

Table C-11 – Test cases for Mains Power Supply functionality of TSS

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n6n711}	GTH simulating engine running - On GTH panel insert MAINS OFF and no commands for 3 min	Respective yellow warning illuminates on MEOCS panel – After 3 min elapsed time system shutdown – RMI server ready to accept connection
2.	{ n1n2n3n4n6n8n9n10n11}	GTH simulating engine running - On GTH panel insert MAINS OFF and no commands for 2.5 min – After 2.5 min hit exit on MEOCS panel	Respective yellow warning illuminates on MEOCS panel – After 2.5 min and prior to 3 min system shutdown – RMI server ready to accept connection due to user action
3.	{n1n2n3n5n9n10n11}	GTH simulating engine running - On GTH panel insert MAINS ON and no commands for 3 min – After 3 min hit exit on MEOCS pan	System shutdown – RMI server ready to accept connection due to user action

C.2.4 General Test Harness (GTH)

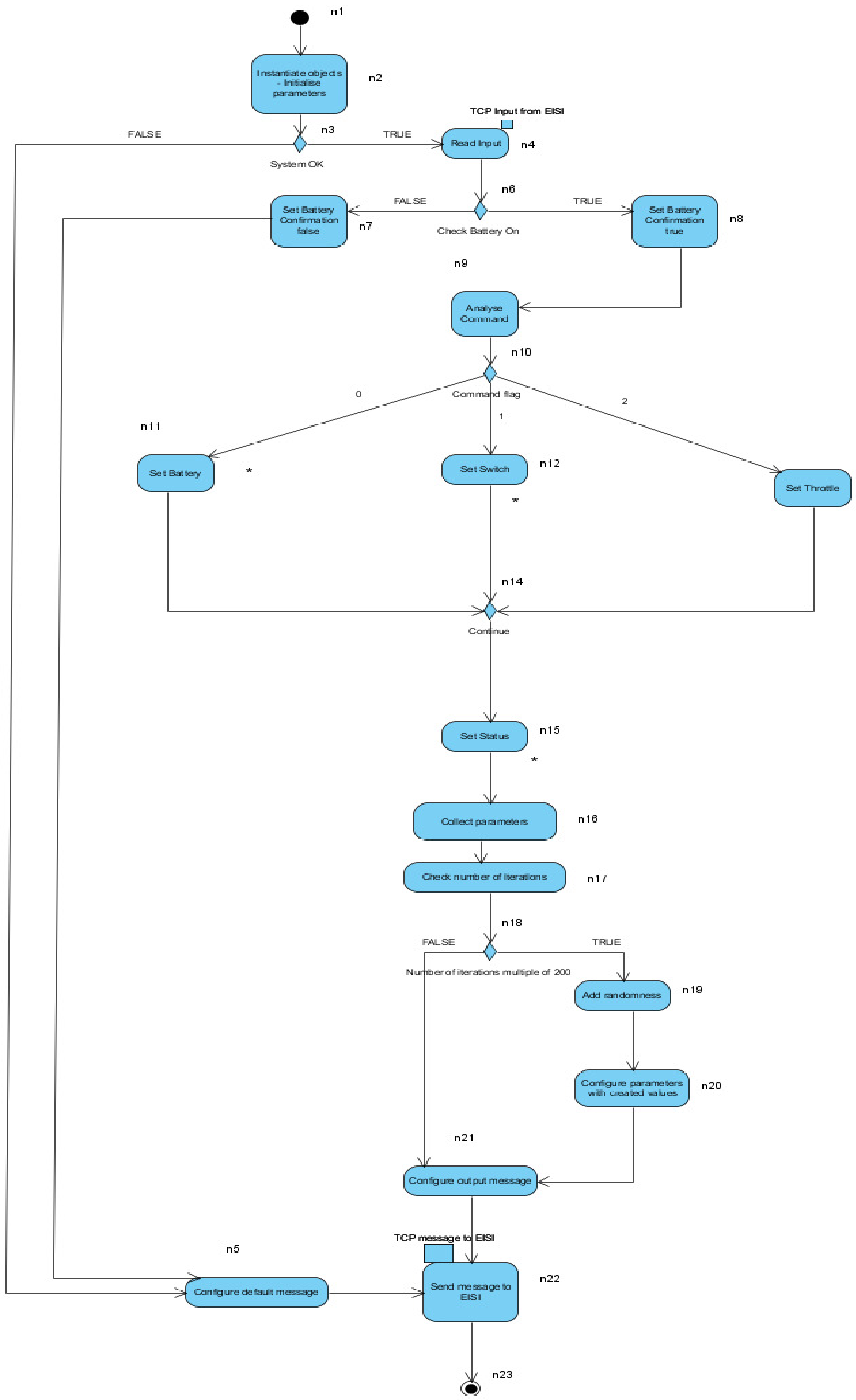


Figure C-11 – General Test Harness control flow

Nodes: n = 23
Edges: e = 27
Cyclomatic Complexity:
VG = e – n + 2 = 27 – 23 + 2 = 6
Individual flow paths: 6

Table C-12 – Test cases for General Test Harness (GTH)

Test Case	Path	Description	Assertions
1.	{n1n2n3n5n22n23}	Click on SYSTEM toggle button	SYSTEM NOT READY indication – all readings 0
2.	{n1n2n3n4n6n7n5n22n23}	SYSTEM ON – send request with flag 001.00 or 002.00 from TCPClient	Battery indication OFF – No other indications
3.	{n1n2n3n4n6n8n9n10n11n14n15n16n17n18n21n22n23}	SYSTEM ON – send request 000.00001.00 from TCPClient	Battery indication ON – Readings indications change appropriately
4.	{n1n2n3n4n6n8n9n10n12n14n15n16n17n18n21n22n23}	SYSTEM ON – send request 000.00001.00 from TCPClient - send request with 001.00 from TCPClient	Switch Indication changes accordingly
5.	{n1n2n3n4n6n8n9n10n13n14n15n16n17n18n21n22n23}	SYSTEM ON – send request 000.00001.00 from TCPClient - send request with 002.00 from TCPClient	Throttle Indication and other indications change accordingly
6.	{n1n2n3n4n6n8n9n10n13n14n15n16n17n18n19n20n21n22n23}	SYSTEM ON – send request 000.00001.00 from TCPClient - send request with 002.00 from TCPClient	Throttle Indication and other indications change accordingly – randomness added to RPM and EGT (they differ from nominal value for that setting)

C.3 JavaScript client (interface.js) functions

C.3.1 Control buttons functionality

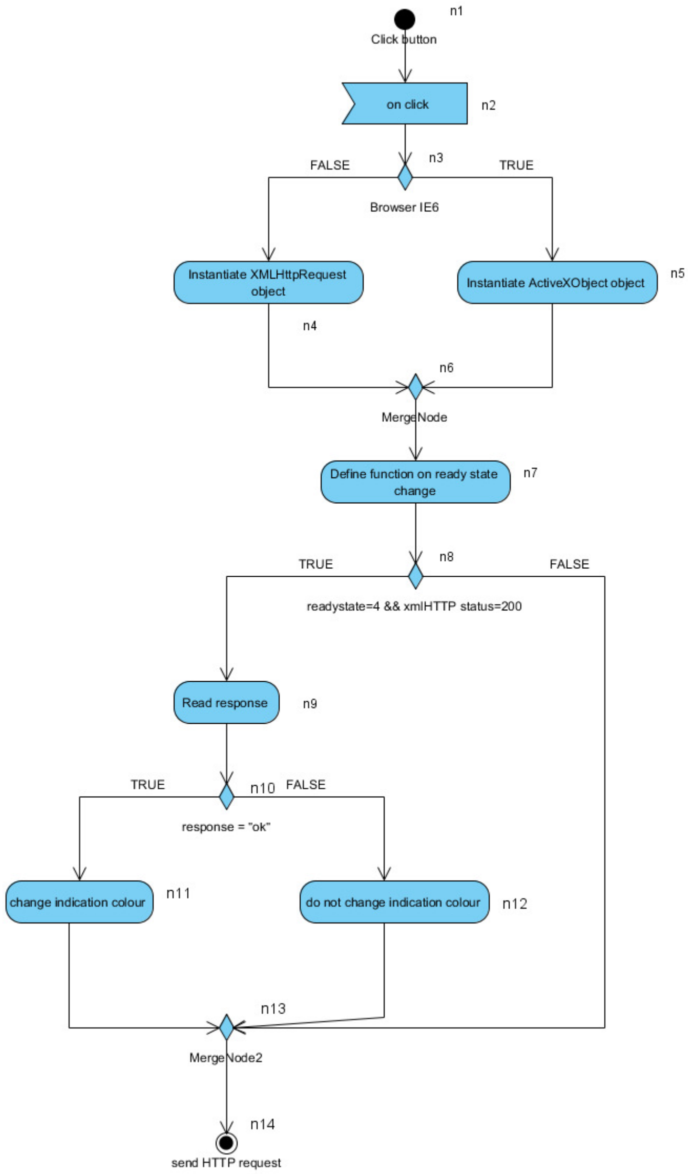


Figure C-12 – Control buttons functionality – control flow

Prerequisites:

1. Web page with YUI buttons running
2. Web console of browser open
3. Test cases 1-3 with IE7 or later, Firefox, Chrome, Opera, Safari
4. Test case 4, 5 with IE6 or earlier

Nodes: n = 14
Edges: e = 17
Cyclomatic Complexity:
VG = e – n + 2 = 17 – 14 + 2 = 5
Individual flow paths: 5

Table C-13 – Test cases for Control buttons functionality of interface.js

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n6n7n8n9n10n11n13n14}	Click on tested button	On web console request* is shown – On specified field colour changed to green and in 2 seconds back to whitesmoke
2.	{n1n2n3n4n6n7n8n9n10n12n13n14}	On server side change return in CommandResource root methods to 'not ok' – click on tested button	On web console request* is shown – On specified field colour does not change
3.	{ n1n2n3n4n6n7n8n13n14}	Distort connection with server – click on tested button – repeat click	On web console status !=200 – no further requests are transmitted on click
4.	{n1n2n3n5n6n7n8n9n10n11n13n14}	Click on tested button	On web console request* is shown – On specified field colour changed to green and in 2 seconds back to whitesmoke
5.	{n1n2n3n5n6n7n8n9n10n12n13n14}	On server side change return in CommandResource root methods to 'not ok' – click on tested button	On web console request* is shown – On specified field colour does not change

* Battery On button request form: "POST","/InternetGasturbine/webresources/command/1/1"
Battery Off button request form: "POST","/InternetGasturbine/webresources/command/1/0"
Stop button request form: "POST","/InternetGasturbine/webresources/command/2/0"
AutoStop button request form: "POST","/InternetGasturbine/webresources/command/2/1"
Run button request form: "POST","/InternetGasturbine/webresources/command/2/2"

C.3.2 Throttle slider functionality

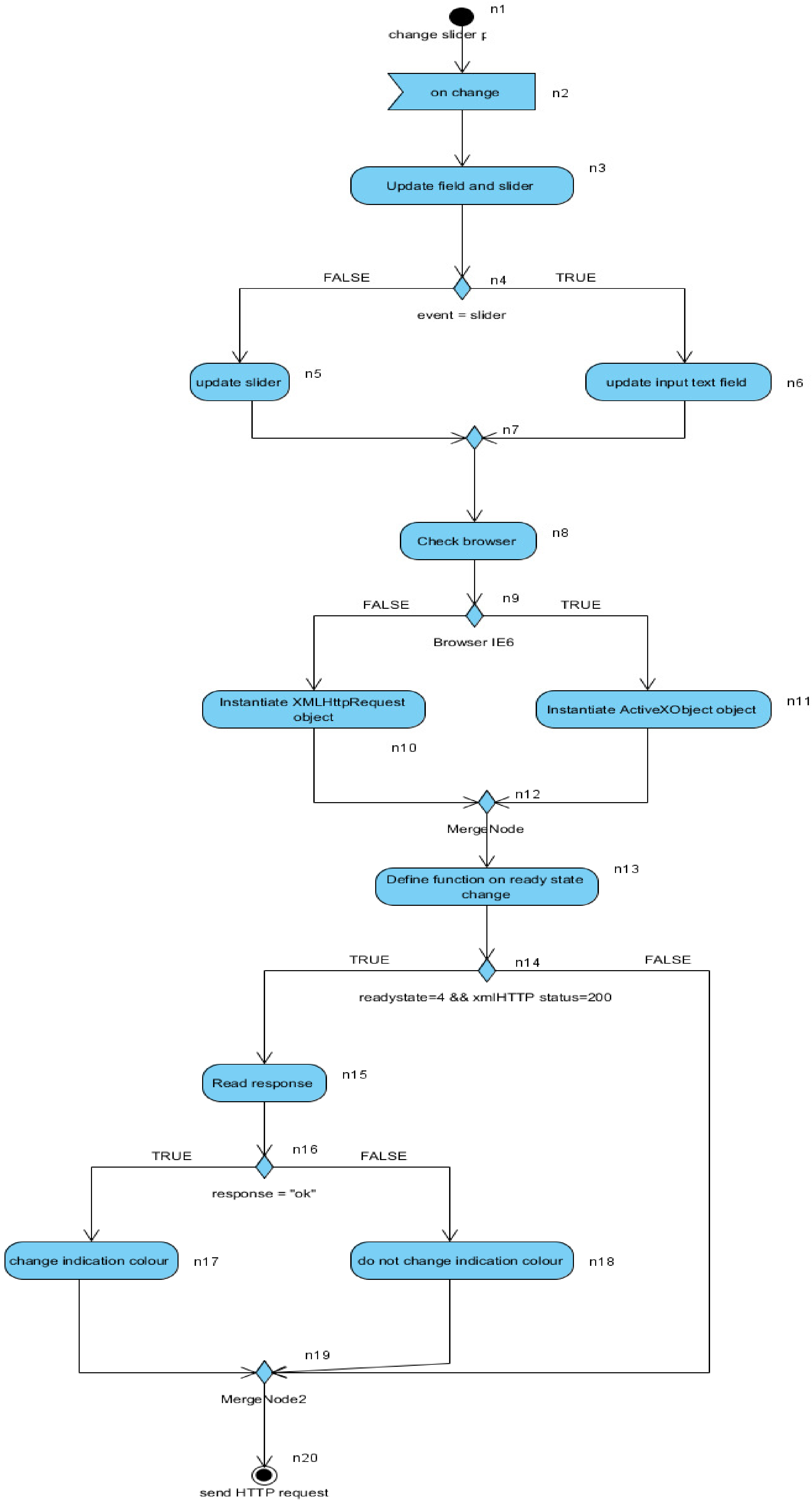


Figure C-13 – Throttle slider functionality control flow

Prerequisites:

- Web page with YUI slider running
- Web console of browser open
- Test cases 1-4 with IE7 or later, Firefox, Chrome, Opera, Safari
- Test case 5 with IE6 or earlier

Nodes: n = 20
Edges: e = 23
Cyclomatic Complexity:
VG = e – n + 2 = 23 – 20 + 2 = 5
Individual flow paths: 5

Table C-14 – Test cases for Throttle Slider functionality

Test Case	Path	Description	Assertions
1.	{n1n2n3n5n7n8n9n10n12n13n14n15n16n17n19n20}	Move slider pointer	On web console request* is shown – On specified field colour changed to green and in 2 seconds back to whitesmoke – slider text field updated accordingly
2.	{n1n2n3n6n7n8n9n10n12n13n14n15n16n17n19n20}	Change setting in throttle text field	On web console request* is shown – On specified field colour changed to green and in 2 seconds back to whitesmoke – slider updated accordingly
3.	{n1n2n3n5n7n8n9n10n12n13n14n15n16n18n19n20}	On server side change return in CommandResource root methods to 'not ok' – Move slider pointer	On web console request* is shown – On specified field colour does not change
4.	{n1n2n3n5n7n8n9n10n12n13n14n19n20}	Distort connection with server – Move slider pointer – repeat move slider pointer	On web console status !=200 – no further requests are transmitted on click
5.	{n1n2n3n5n7n8n9n11n12n13n14n15n16n17n19n20}	Move slider pointer	On web console request* is shown – On specified field colour changed to green and in 2 seconds back to whitesmoke – slider text field updated accordingly

* Slider request form: "POST","/InternetGasturbine/webresources/command/3/{integer rang 0-100}"

C.3.3 Quit execution function (quitPage)

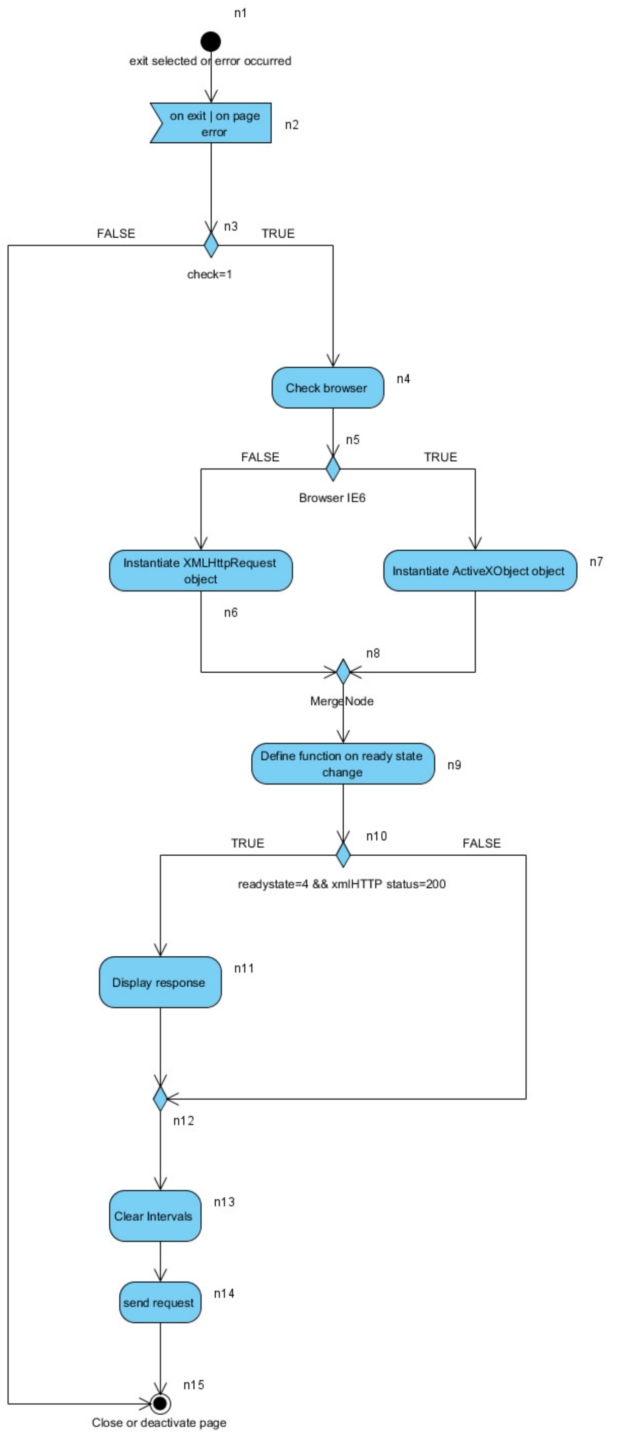


Figure C-14 – Quit execution function (quitPage) control flow

Prerequisites:

Check point 1: printout 'request received'at StopResource() root methods

- Check point 2: printout 'request received'at readAmbResource() root methods
- Check point 3: printout 'request received'at readEngResource() root methods
- readAmb() and reading() functions already embedded
- Web console of browser open
- Test cases 1-3 with IE7 or later, Firefox, Chrome, Opera, Safari
- Test case 4 with IE6 or earlier

Nodes: n = 15

Edges: e = 17

Cyclomatic Complexity:

VG = e – n + 2 = 17 – 15 + 2 = 4

Independent flow paths: 4

Table C-15 – Test cases for Quit execution function (quitPage)

Test Case	Path	Description	Assertions
1.	{n1n2n3n15}	Modify variable check on page source to be check = 1 – select Exit	On web console no request* is shown – On server side no request confirmation is printed at check point 1
2.	{n1n2n3n4n5n6n8n9n10n11n12n13n14n15}	select Exit	On web console request* is shown – Page changes theme prior to closure – no requests confirmation printouts at check points 2 and 3
3.	{n1n2n3n4n5n6n8n9n10}	Distort connection with server	On web console request* is shown – Page changes theme prior to closure – no requests confirmation printouts at check points 2 and 3
4.	{n1n2n3n4n5n7n8n9n10n11n12n13n14n15}	select Exit	On web console request* is shown – Page changes theme prior to closure – no requests confirmation printouts at check points 2 and 3

* Request form: "GET","/InternetGasturbine/webresources/stop"

C.3.4 Logout function (logoutPage)

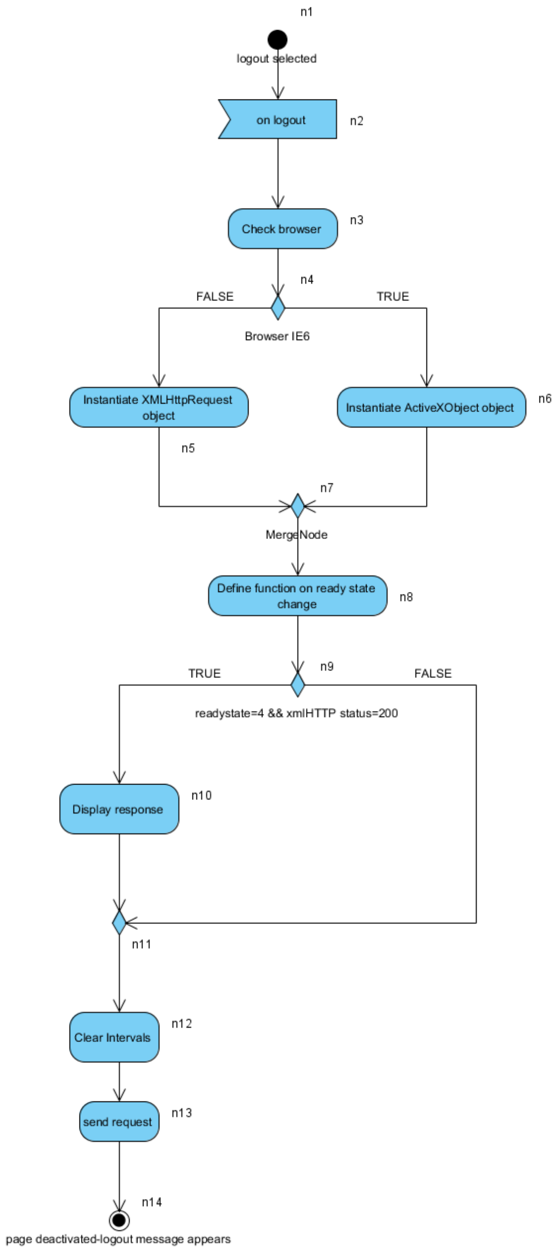


Figure C-156 – Logout function (logoutPage) control flow

Prerequisites:

- Check point 4: printout ‘request received’at LogoutResource() root methods
- Check point 2: printout ‘request received’at readAmbResource() root methods
- Check point 3: printout ‘request received’at readEngResource() root methods
- readAmb() and reading() functions already embedded

- Web console of browser open
- Test cases 1,2 with IE7 or later, Firefox, Chrome, Opera, Safari
- Test case 3 with IE6 or earlier

Nodes: n = 14
Edges: e = 15
Cyclomatic Complexity:
VG = e – n + 2 = 15 – 14 + 2 = 3
Independent flow paths: 3

Table C-16 – Test cases for logout function (logoutPage)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n5n7n8n9n10n11n12n13n14}	Select logout	On web console request* is shown – Request confirmation printout at check point 4 - Page changes theme – no requests confirmation printouts at check points 2 and 3
2.	{n1n2n3n4n5n7n8n9n11n12n13n14	Distort connection with server – select logout	On web console request* is shown – error message displayed – no requests confirmation printouts at check points 2 and 3 and 4
3.	{n1n2n3n4n6n7n8n9n10n11n12n13n14}	Select logout	On web console request* is shown – Request confirmation printout at check point 4 - Page changes theme – no requests confirmation printouts at check points 2 and 3

* Request form: "GET","/InternetGasturbine/webresources/logout"

C.3.5 Ambient data and TSS warnings acquisition function

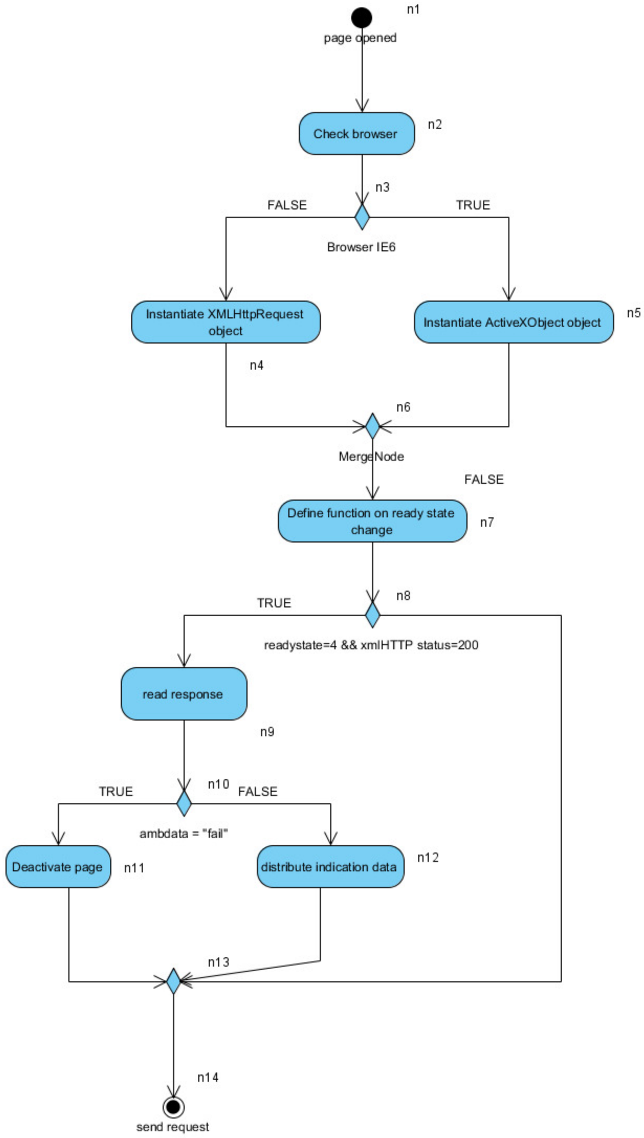


Figure C-167 – Ambient data and TSS warnings acquisition function (readAmb) control flow

Prerequisites:

- Check point 2: printout 'request received'at readAmbResource() root methods
- Check point 3: printout 'request received'at readEngResource() root methods
- All other functions already embedded
- Web console of browser open
- Test cases 1-3 with IE7 or later, Firefox, Chrome, Opera, Safari
- Test case 4 with IE6 or earlier

Nodes: n = 14
Edges: e = 16
Cyclomatic Complexity:
VG = e – n + 2 = 16 – 14 + 2 = 4
Independent flow paths: 4

Table C-18 – Test cases for ambient data and TSS warnings acquisition function (readAmb) control flow

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n6n7n8n9n10n12n13n14}	Open page	On web console request* is shown – Ambient parameters indications are displayed in fields as expected – requests confirmation printouts at check points 2 and 3
2.	{n1n2n3n4n5n6n8n9n10n10n12n13n14n15}	Modify ReadAmbResource() root methods to return 'fail'	On web console request* is shown – Page shows error message – no requests shown in web console - no requests confirmation printouts at check points 2 and 3
3.	{n1n2n3n4n5n6n8n9n10}	Distort connection with server	On web console request* is shown – Page shows error message – no requests shown in web console - no requests confirmation printouts at check points 2 and 3
4.	{n1n2n3n4n6n7n8n9n10n12n13n14}	Open page	On web console request* is shown – Ambient parameters indications are displayed in fields as expected – requests confirmation printouts at check points 2 and 3

* Request form: "POST","/InternetGasturbine/webresources/readAmb"

C.3.6 Engine data and system status acquisition function

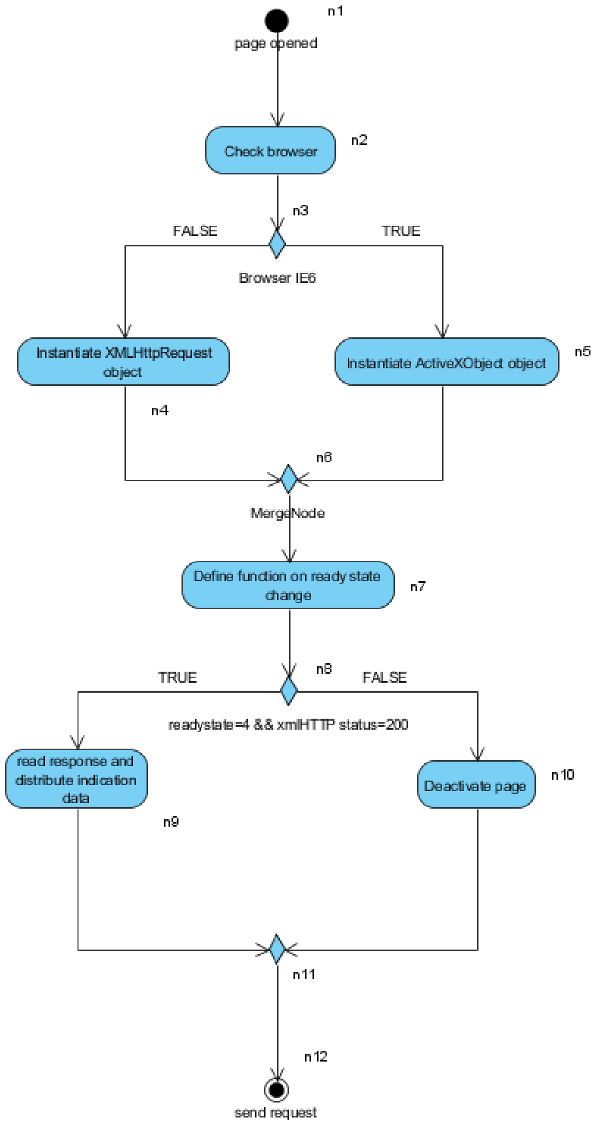


Figure C-17 – Engine data and system status acquisition function (readEng) control flow

Prerequisites:

- Check point 2: printout ‘request received’at readAmbResource() root methods
- Check point 3: printout ‘request received’at readEngResource() root methods
- All other functions already embedded
- Web console of browser open
- Test cases 1, 2 with IE7 or later, Firefox, Chrome, Opera, Safari
- Test case 3 with IE6 or earlier

Nodes: n = 14
Edges: e = 16
Cyclomatic Complexity:
VG = e – n + 2 = 16 – 14 + 2 = 4
Independent flow paths: 4

Table C 19 – Test cases for engine data and system status acquisition function (readEng)

Test Case	Path	Description	Assertions
1.	{n1n2n3n4n6n7n8n9n11n12}	Open page	On web console request* is shown – Engine parameters indications are displayed in fields as expected – requests confirmation printouts at check points 2 and 3
2.	{n1n2n3n4n6n7n8n10n11n12}	Distort connection with server	On web console request* is shown – Page shows error message – no requests shown in web console - no requests confirmation printouts at check points 2 and 3
3.	{n1n2n3n4n6n7n8n9n11n12}	Open page	On web console request* is shown – Engine parameters indications are displayed in fields as expected – requests confirmation printouts at check points 2 and 3

* Request form: "POST","/InternetGasturbine/webresources/readEng"

Appendix D Acceptance and Final Validation

D.1 State machines and NuSMV models

D.1.1 State machine SM0

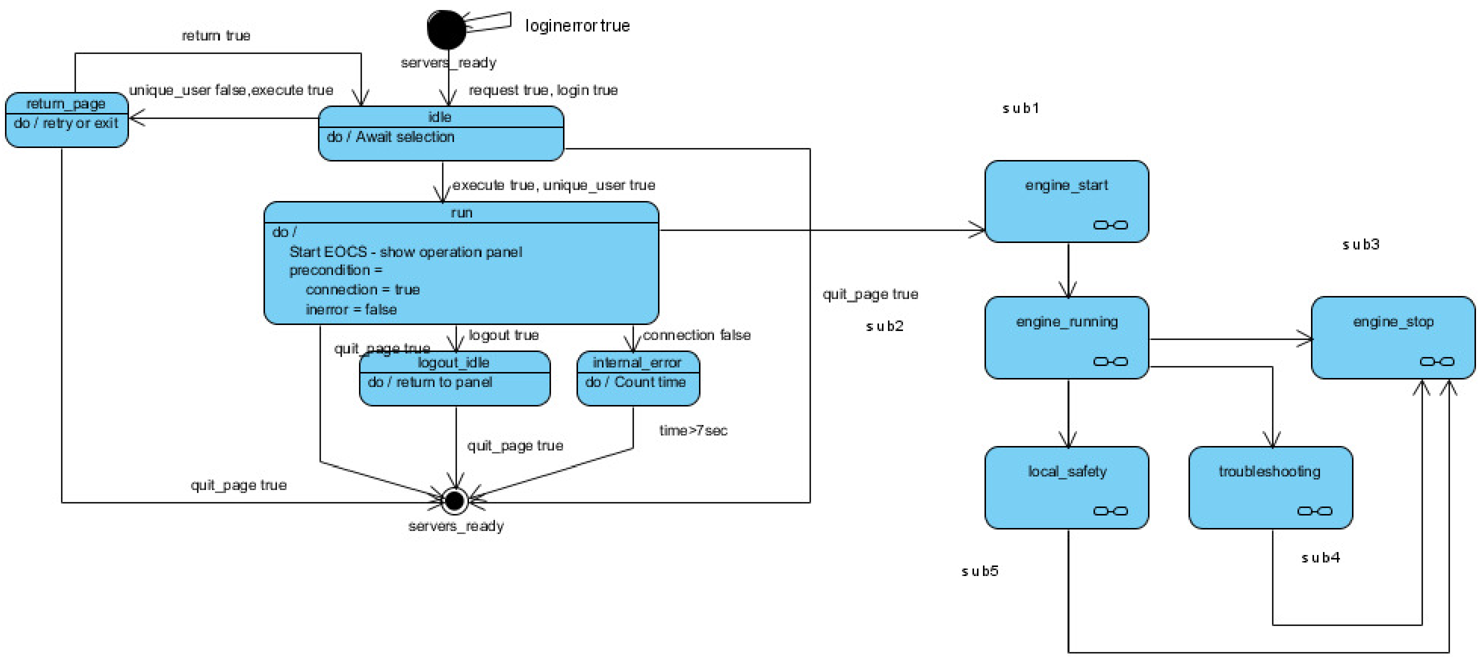


Figure D-1 - State machine of main web application (SM0)

The NuSMV model for sm0 contains a main module that includes 1 state variable (VAR) *state* and 9 input variables (IVAR) that represent all the potential inputs from the user. By initially assigning the state variable *state* to the value *ready,* all the possible next values were exhaustively defined according to the pervious value of the state variable and the user input (Figure D-2).

```
MODULE main
VAR
 state:{ready,idle,run,return_page,logout_idle,internal_error};
IVAR
 loginerror : boolean;
 unique_user:boolean;
 login:boolean;
 logout:boolean;
 request:boolean;
 execute:boolean;
 quit_page:boolean;
 interror:boolean;
 connection:boolean;
ASSIGN
 init(state):=ready;
 next(state):= case
                        state=ready & logout=FALSE & login=TRUE & request=TRUE & loginerror=TRUE:ready;
                        state=ready & logout=FALSE & login=TRUE & request=TRUE & loginerror=FALSE:idle;
                        state=ready & quit_page=TRUE:ready;
                        state=idle & login=TRUE & logout=FALSE & execute=TRUE & unique_user=FALSE :return_page;
                        state=idle & logout=FALSE & execute=TRUE & unique_user=TRUE & connection=TRUE:run;
                        state=idle & login=TRUE & logout=FALSE & quit_page=TRUE:ready;
                        state=run & unique_user=TRUE & logout=FALSE & connection=FALSE:internal_error;
                        state=run & unique_user=TRUE & logout=FALSE & interror=TRUE:internal_error;
                        state=run & unique_user=TRUE & logout=TRUE:logout_idle;
                        state=run & unique_user=TRUE & quit_page=TRUE:ready;
                        state=logout_idle & quit_page=TRUE:ready;
                        state=return_page & quit_page=TRUE:ready;
                        state=internal_error & quit_page=TRUE:ready;
                        TRUE:state;
                     esac;
```

Figure D-2 - NuSMV model for main state machine sm0 of the application

There were 11 CTL specifications defined upon the previous NuSMV model. They involved AG and EX formulae implying global investigation for transition from one state to the next (Figure D-3). Had these specifications not been negated, they would have been evaluated true. However, the negation leads the program to produce counter examples showing the potential test cases (Figure D-4Figure D-4 – Counterexample for specification 0.5 of NuSMV model of sm0).

```
-- spec 0.1
CTLSPEC
AG(state=ready -> !EX state=ready)

-- spec 0.2
CTLSPEC
AG(state=ready -> !EX state=idle)

-- spec 0.3
CTLSPEC
AG(state=idle -> !EX state=run)

-- spec 0.4
CTLSPEC
AG(state=idle -> !EX state=return_page)

-- spec 0.5
CTLSPEC
AG(state=idle -> !EX state=ready)

-- spec 0.6
CTLSPEC
AG(state=run -> !EX state=ready)

-- spec 0.7
CTLSPEC
AG(state=run -> !EX state=internal_error)

-- spec 0.8
CTLSPEC
AG(state=run -> !EX state=logout_idle)

-- spec 0.9
CTLSPEC
AG(state=logout_idle -> !EX state=ready)

-- spec 0.10
CTLSPEC
AG(state=return_page -> !EX state=ready)

-- spec 0.11
CTLSPEC
AG(state=internal_error -> !EX state=ready)
```

Figure D-3 – CTL specifications for NuSMV model of sm0

```
-- specification AG (state = idle -> !(EX state = ready))  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 5.1 <-
  state = ready
-> Input: 5.2 <-
  loginerror = FALSE
  unique_user = FALSE
  login = TRUE
  logout = FALSE
  request = TRUE
  execute = FALSE
  quit_page = FALSE
  interror = FALSE
  connection = FALSE
-> State: 5.2 <-
  state = idle
-> Input: 5.3 <-
  loginerror = FALSE
  unique_user = TRUE
  login = TRUE
  logout = FALSE
  request = FALSE
  execute = TRUE
  quit_page = TRUE
  interror = FALSE
  connection = FALSE
-> State: 5.3 <-
  state = ready
```

Figure D-4 – Counterexample for specification 0.5 of NuSMV model of sm0

D.1.2 State machine SM1

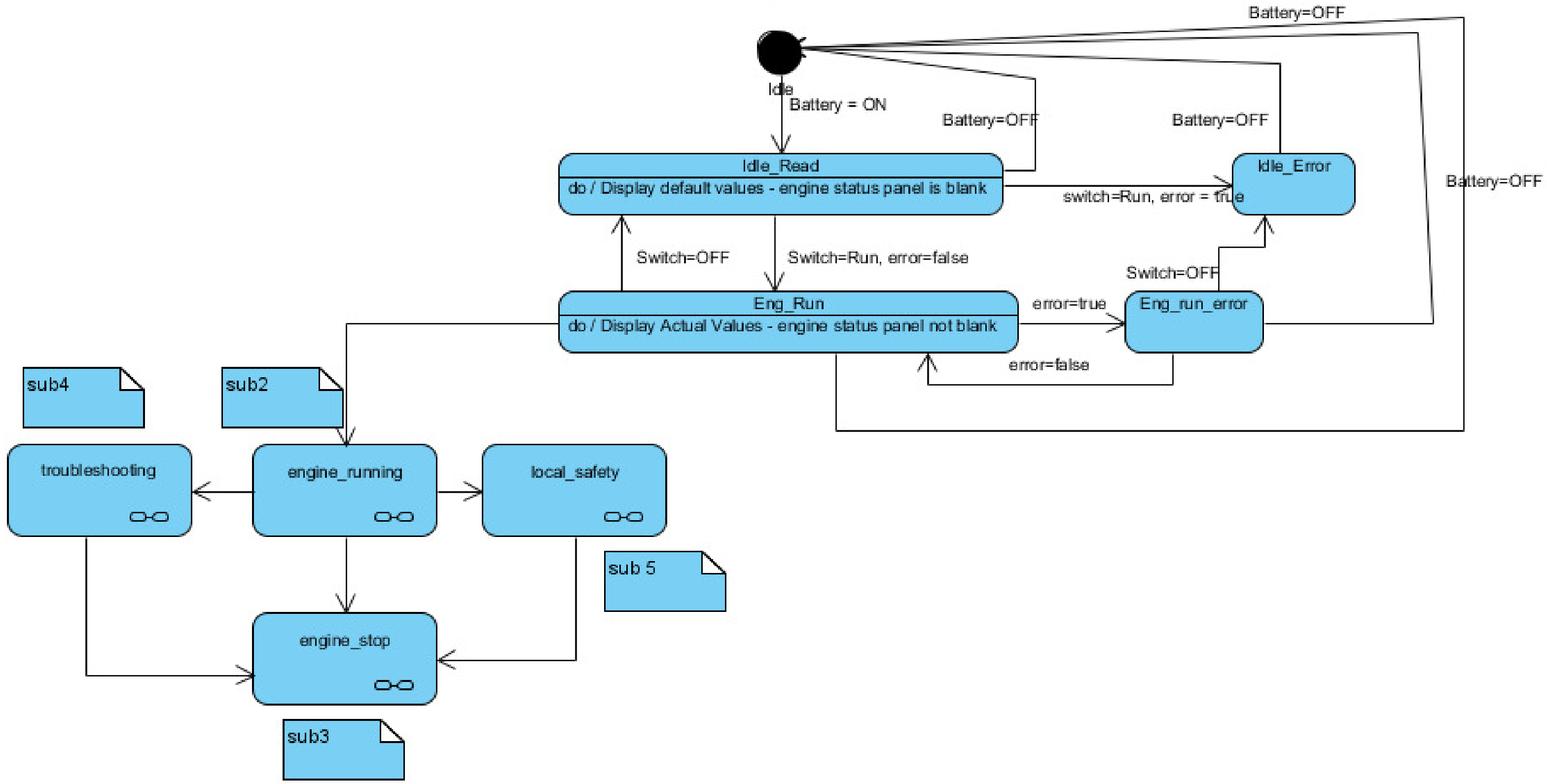


Figure D-5 - State machine of engine start sub state (SM1)

The same approach was followed for sm1, with AG and AX formulae to define ten CTL specifications (Figure D-7). The model in this case was more complicated, as there was a combination of state variables to describe the transitional states (Figure D-6). The model was attempted to be designed as a descriptive abstraction of the actual variables involved in the start modes of the application. The start state is identical to the end state, which represents the program running but the gas turbine not.

```
MODULE main
VAR
 state:{idle,idle_read,idle_error,eng_run, eng_run_error};
 rpm:{zero,actual};
 egt:{default,actual};
 throtfb:{zero,actual};
 vsup:{zero,actual};
 vout:{zero,actual};
 engine_status_panel:{blank,read};
 engine_warning_panel:{read,blank};
 system_status_panel:{read};
 ambient:{actual};

IVAR
 switch:{off-auto,run};
 battery:{off,on};
 error:boolean;


ASSIGN
 init(state):=idle;
 init(rpm):=zero;
 init(egt):=default;
 init(throtfb):=zero;
 init(vsup):=zero;
 init(vout):=zero;
 init(engine_status_panel):=blank;
 init(engine_warning_panel):=blank;
```

Figure D-6 - NuSMV model for sm1 of the application

```
--Spec 1.1
CTLSPEC
AG(state=idle -> !EX state=idle_read)

--Spec 1.2
CTLSPEC
AG(state=idle_read -> !EX state=idle_error)

--Spec 1.3
CTLSPEC
AG(state=idle_read -> !EX state=eng_run)

--Spec 1.4
CTLSPEC
AG(state=eng_run -> !EX state=eng_run_error)

--Spec 1.5
CTLSPEC
AG(state=eng_run -> !EX state=idle)

--Spec 1.6
CTLSPEC
AG(state=idle_error -> !EX state=idle)

--Spec 1.7
CTLSPEC
AG(state=eng_run_error -> !EX state=idle)

--Spec 1.8
CTLSPEC
AG(state=eng_run -> !EX state=idle_read)

--Spec 1.9
CTLSPEC
AG(state=eng_run_error -> !EX state=idle_error)

--Spec 1.10
CTLSPEC
AG(state=idle_read -> !EX state=idle)
```

Figure D-7 - CTL specifications for NuSMV model of sm1

D.1.3 State Machine SM2

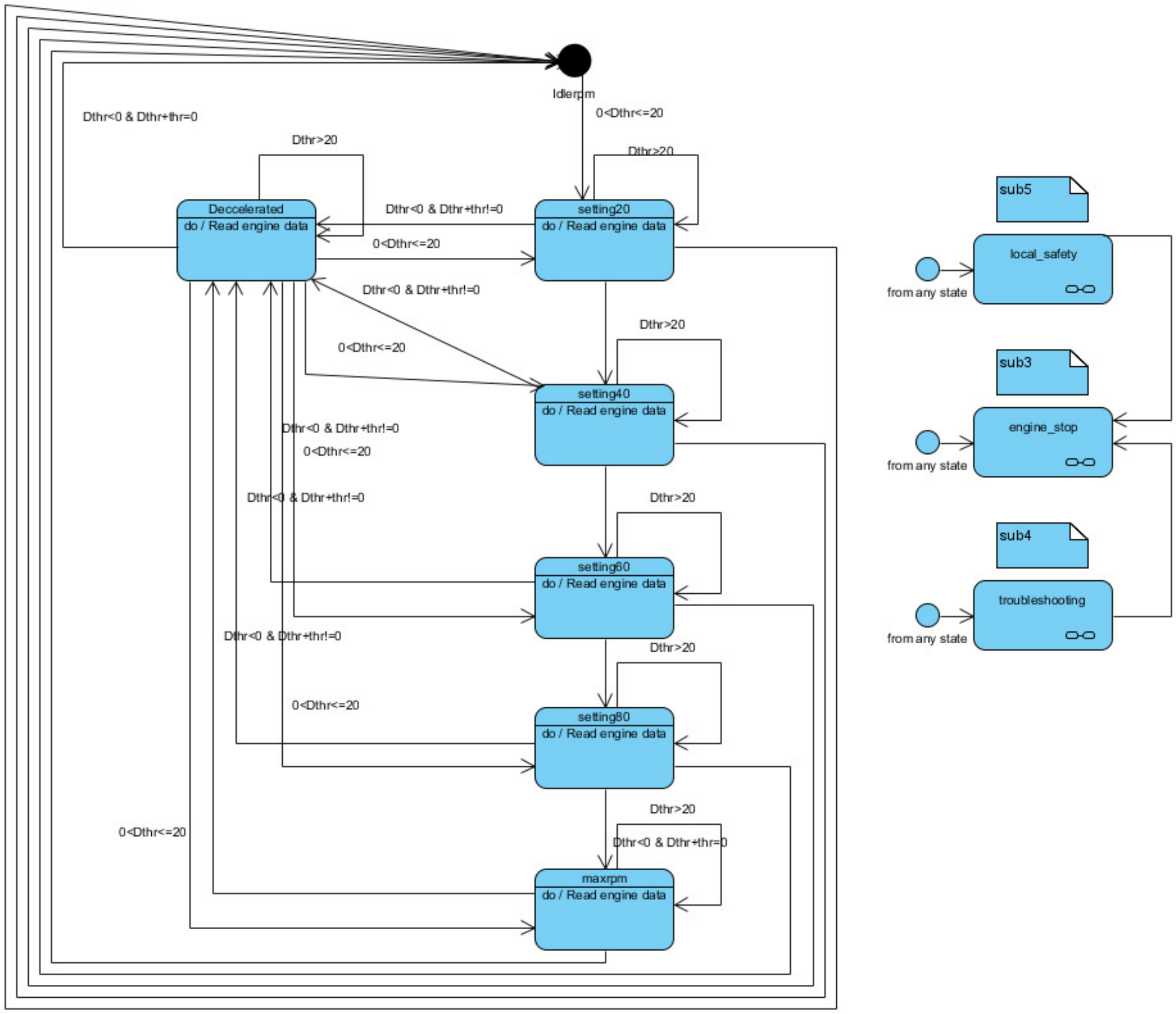


Figure D-8 - State machine of engine running sub state (SM2)

The model corresponding to *sm2* (Figure D-8) applied 2 approaches for the definition of the specifications (Figure D-9). The main one was identical to the previous models, using CTL with AG and EX formulae. With the throttle input as an input variable and a combination of state variables, the model navigated throughout the whole envelope of the throttle settings, being advanced in intervals of 20%. The second approach applied LTL to show that the model would not allow direct advance from a state where the engine and the related indications correspond to idle throttle, to a state that represents 100% throttle. In the last case, the LTL counterexample indicated that such a transition would be possible only by intervals of 20% throttle increase maximum.

```
MODULE main
VAR
 thr:0..100;
 egt:395..595;
 rpm:45..145;
 vout:0..10;
 fflow:5..305;
 idlerpm:boolean;
 setting20:boolean;
 setting40:boolean;
 setting60:boolean;
 setting80:boolean;
 maxrpm:boolean;
IVAR
 dthr:-100..100;
ASSIGN
 init(rpm):=45;
 init(egt):=395;
 init(fflow):=5;
 init(thr):=0;
 init(vout):=0;
```

Figure D-9 - NuSMV model for SM2

```
-- spec 2.1
LTLSPEC
G(X !(rpm = 145 & egt = 595 & fflow = 305 & thr = 100 & vout = 10 & maxrpm = TRUE))

-- spec 2.2
CTLSPEC
AG((idlerpm = FALSE & maxrpm = TRUE)-> !EX idlerpm = TRUE)

-- spec 2.3
CTLSPEC
AG((idlerpm = FALSE & setting20 = TRUE) -> !EX(idlerpm = TRUE))

-- spec 2.4
CTLSPEC
AG((idlerpm = FALSE & setting40 = TRUE) -> !EX(idlerpm = TRUE))

-- spec 2.5
CTLSPEC
AG((idlerpm = FALSE & setting60 = TRUE) -> !EX(idlerpm = TRUE))

-- spec 2.6
CTLSPEC
AG((idlerpm = FALSE & setting80 = TRUE) -> !EX(idlerpm = TRUE))

-- spec 2.7
CTLSPEC
AG((setting40 = FALSE & setting80 = TRUE) -> !EX(setting40 = TRUE))

-- spec 2.8
CTLSPEC
AG((setting60 = TRUE & setting80 = FALSE) -> !EX(setting80 = TRUE))

-- spec 2.9
CTLSPEC
AG((setting40 = TRUE & setting80 = FALSE) -> !EX(setting80 = TRUE))
```

Figure D-10 – LTL and CTL specifications for NuSMV model of SM2

D.1.4 State machine SM3

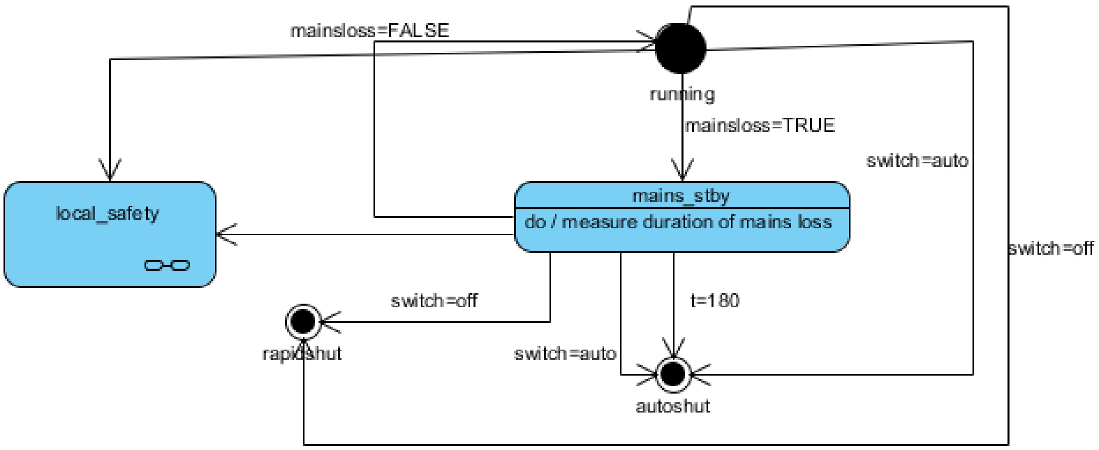


Figure D-11 - State machine of engine stop and mains power supply monitoring sub state (SM3)

The model of sm3 (Figure D-12) represents the normal shutdown of the engine along with automatic shutdown from the program based on power supply loss. This model applies the mostly common herein approach with CTL and, AG and EX formulae for the definition of 7 specifications (Figure D-13). However, one more specification was defined with the use of AG and EF formulae. This was expected to produce a counterexample that if the engine was running there could be a series of transitions after which eventually the system would be in the state of rapid shutdown, which represents the actual STOP condition of the EDT. Time can also be seen here as a state variable. Certain values of time in certain states define a transition to a next one.

```
MODULE main
VAR
 state:{running,mains_stby,rapidshut,autoshut};
 t:0..180;
IVAR
 switch:{on,off,auto};
  mainsloss:boolean;
ASSIGN
 init(state):=running;
 init(t):=0;
```

Figure D-12 - NuSMV model for SM3

```
-- spec 3.1
CTLSPEC
AG(state=running -> !EX state=running)

-- spec 3.2
CTLSPEC
AG(state=running -> !EX state=mains_stby)

-- spec 3.3
CTLSPEC
AG(state=running -> !EF state=rapidshut)

-- spec 3.4
CTLSPEC
AG(state=running -> !EX state=autoshut)

-- spec 3.5
CTLSPEC
AG(state=mains_stby -> !EX state=running)

-- spec 3.6
CTLSPEC
AG(state=mains_stby -> !EX state=rapidshut)

-- spec 3.7
CTLSPEC
AG(state=mains_stby & t!=180 -> !EX state=autoshut)

-- spec 3.8
CTLSPEC
AG(state=mains_stby & t=180 -> !EX state=autoshut)
```

Figure D-13 - CTL specifications for NuSMV model of SM3

D.1.5 State machine SM4

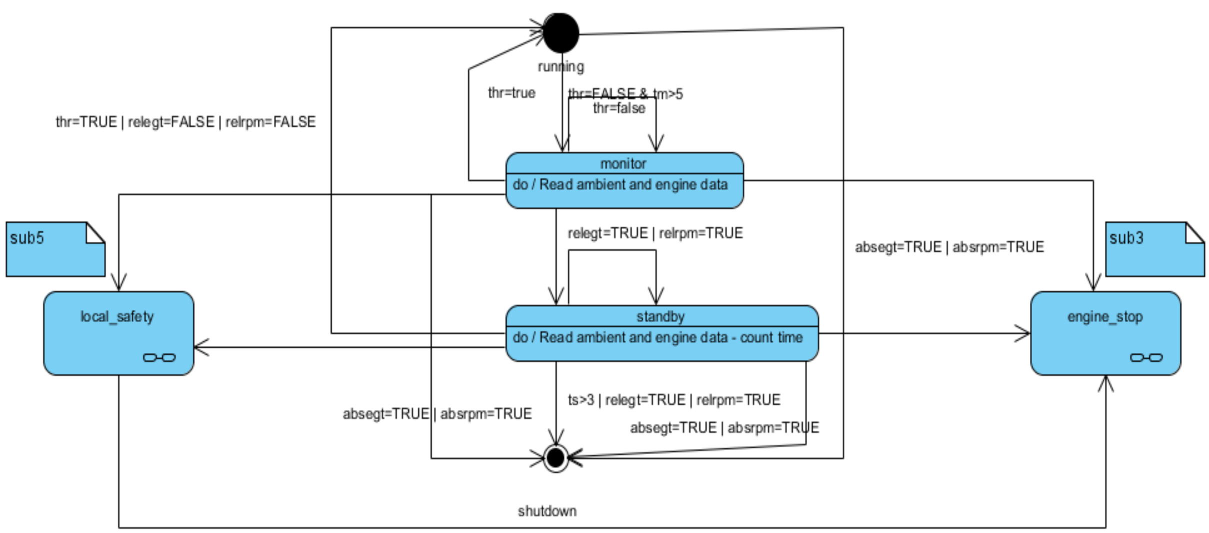


Figure D-14 - State machine of trouble shooting sub state (SM4)

The model sm4 (Figure D-15) applied CTL logic with AG and EX formulae and defined 12 specifications (Figure D-16). Time was also declared as a state variable and particularly two different time variables were defined. t_m was used to simulate the time elapsed since the last throttle movement. If $t_m \geq 5$ the program enters monitoring state. Variable t_s is the time spent whilst in relative warning standby state. If $t_s \geq 180$ the program enters shutdown state.

```
MODULE main
VAR
 state:{running,monitor,stby,shutdown};
 input:{normal,relind,absind};
 tm:0..5;
 ts:0..180;
IVAR
 thr:boolean;
ASSIGN
 init(state):=running;
 init(input):=normal;
```

Figure D-15 - NuSMV model for sm4 of the application

```
-- spec 4.1
CTLSPEC
AG(state=running -> !EX state=running)

-- spec 4.2
CTLSPEC
AG(state=running -> !EX state=monitor)

-- spec 4.3
CTLSPEC
AG(state=running -> !EX state=stby)

-- spec 4.4
CTLSPEC
AG(state=running -> !EX state=shutdown)

-- spec 4.5
CTLSPEC
AG(state=monitor -> !EX state=stby)

-- spec 4.6
CTLSPEC
AG(state=monitor -> !EX state=shutdown)

-- spec 4.7
CTLSPEC
AG(state=monitor -> !EX state=running)

-- spec 4.8
CTLSPEC
AG(state=stby & input!=absind -> !EX state=shutdown)

-- spec 4.9
CTLSPEC
AG(state=stby & input=absind -> !EX state=shutdown)

-- spec 4.10
CTLSPEC
AG(state=stby -> !EX state=running)

-- spec 4.11
CTLSPEC
AG(state=stby & input!=normal & input!=absind -> !EX state=running)

-- spec 4.12
CTLSPEC
AG(state=stby -> !EX state=monitor)
```

Figure D-16 - CTL specifications for NuSMV model of sm4

D.1.6 State machine SM5

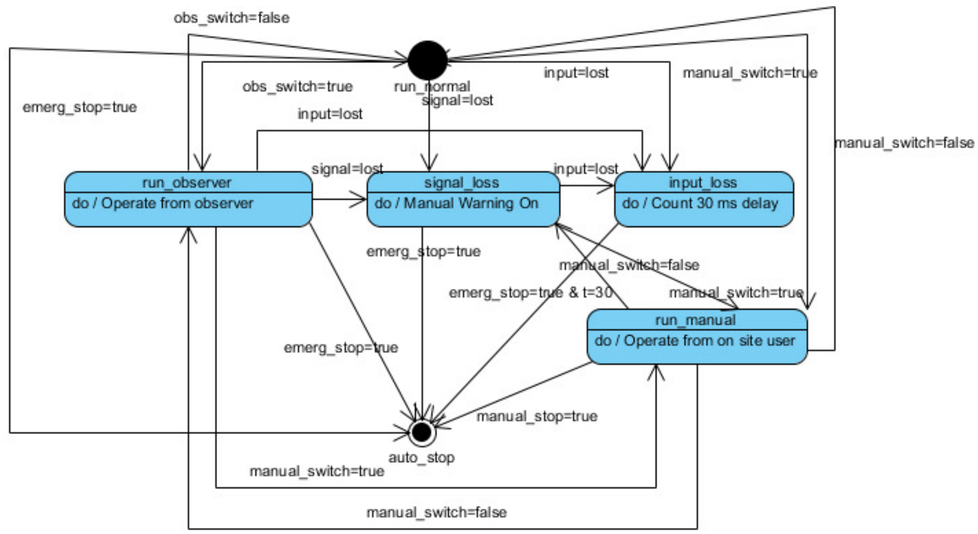


Figure D-17 - State machine of local safety sub state (SM5)

The final model sm5 (Figure D-18) also applied CTL with AG and EX formulae with 18 specifications defined (Figure D-19).

```
MODULE main
VAR
 state:{run_normal,run_observer,signal_loss,input_loss,run_manual,auto-stop};
 manual_warning:{off,on};
 t:0..30;
IVAR
 signal:{valid,lost};
 input:{valid,lost};
 obs_switch:boolean;
 manual_switch:boolean;
 emerg_stop:boolean;
 manual_stop:boolean;
ASSIGN
 init(state):=run_normal;
 init(manual_warning):=off;
```

Figure D-18 - NuSMV model for sm5 of the application

```
-- spec 5.1
CTLSPEC
AG(state=run_normal -> !EX state=auto-stop)

-- spec 5.2
CTLSPEC
AG(state=run_normal -> !EX state=signal_loss)

-- spec 5.3
CTLSPEC
AG(state=run_normal -> !EX state=run_observer)

-- spec 5.4
CTLSPEC
AG(state=run_normal -> !EX state=input_loss)

-- spec 5.5
CTLSPEC
AG(state=run_normal -> !EX state=run_manual)

-- spec 5.6
CTLSPEC
AG(state=run_observer -> !EX state=auto-stop)

-- spec 5.7
CTLSPEC
AG(state=run_observer -> !EX state=signal_loss)

-- spec 5.8
CTLSPEC
AG(state=run_observer -> !EX state=run_normal)

-- spec 5.9
CTLSPEC
AG(state=run_observer  -> !EX state=input_loss)

-- spec 5.10
CTLSPEC
AG(state=run_observer  -> !EX state=run_manual)

-- spec 5.11
CTLSPEC
AG(state=signal_loss -> !EX state=auto-stop)

-- spec 5.12
CTLSPEC
AG(state=signal_loss -> !EX state=run_manual)

-- spec 5.13
CTLSPEC
AG(state=input_loss -> !EX state=auto-stop)

-- spec 5.14
CTLSPEC
AG(state=run_manual -> !EX state=run_normal)

-- spec 5.15
CTLSPEC
AG(state=run_manual -> !EX state=run_observer)

-- spec 5.16
CTLSPEC
AG(state=run_manual -> !EX state=signal_loss)

-- spec 5.17
CTLSPEC
AG(state=run_manual -> !EX state=auto-stop)

-- spec 5.18
CTLSPEC
AG(state=signal_loss -> !EX state=input_loss)
```

Figure D-19 - CTL specifications for NuSMV model of sm5

D.2 Test cases for acceptance validation

The test cases have been grouped in tables that refer to the corresponding state machine. Each row represents a test case with the following attributes: Number of case, NuSMV model specification from which it was derived (format: NuSMV model.spec number), state of the program prerequisite to conduct the case, parameters involved, state to which the program will have transitioned after the conduction of the case and finally, the assertions that need to be verified.

Table D-1 – Test cases derived from SM0

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertion
1	0.1	CTL	ready	loginerror = TRUE	ready	Login error page displayed Servers running and ready to accept
2	0.2	CTL	ready	loginerror = FALSE request = TRUE login = TRUE	idle	Selection page displayed
3	0.3	CTL	idle	unique user = TRUE execute = TRUE connection = TRUE	run	Operation panel displayed Program running
4	0.4	CTL	idle	unique user = FALSE login = TRUE execute = TRUE	return page	Return page displayed instead of Operation panel Program not running
5	0.5	CTL	idle	quit page = TRUE	ready	Selection page closed Program not running Servers ready to accept
6	0.6	CTL	run	quit page = TRUE interror = FALSE connection = TRUE	ready	Operation panel page closed Program not running Servers ready to accept
7	0.7	CTL	run	interror = TRUE connection = FALSE	internal error	Operation panel disappears Error message displayed on page Program not running
8	0.7	CTL	run	interror = TRUE connection = TRUE	Internal error	Operation panel disappears Error message displayed on page Program not running
9	0.7	CTL	run	interror = FALSE connection = FALSE	internal_error	Operation panel disappears Error message displayed on page Program not running
10	0.8	CTL	run	logout = TRUE	logout_idle	Operation panel disappears Logout message displayed on page Program not running

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertion
11	0.9	CTL	logout_idle	quit_page = TRUE	ready	Operation panel page with logout message closes Program not running Servers ready to accept
12	0.10	CTL	return_page	quit_page = TRUE	ready	Return pages closes Program not running Servers ready to accept
13	0.11	CTL	internal_error	quit_page = TRUE	ready	Operation Panel page with error message closes Program not running Servers Ready to accept

Table D-2 – Test cases derived from SM1

Test Case No	**Spec No**	**Spec Type**	**Initial state**	**Parameters**	**Next State**	**Assertions**
1	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = No Serial Input	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : No Serial Input ambient data = actual
2	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Switch signal error	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Switch signal error ambient data = actual
3	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Throttle signal error	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel Throttle signal error ambient data = actual
4	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Switch & throttle	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Switch & throttle ambient data = actual
5	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Switch & serial	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Switch & serial ambient data = actual
6	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Throttle & serial	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Throttle & serial ambient data = actual
7	1.1	CTL	idle	switch = off-auto battery = on Test Harness: Status = Combined	Idle read	rpm : actual egt: actual throttle feedback: zero vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Combined ambient data = actual
8	1.1	CTL	idle	switch = off-auto	Idle read	rpm : actual

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
				battery = on Test Harness: System = Not Ready		egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : System not ready ambient data = actual
9	`1.1	CTL	idle	switch = off-auto battery = on EISI-EIS: Control = Normal	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Combined ambient data = actual Control status: Normal
10	1.1	CTL	idle	switch = off-auto battery = on EISI-EIS: Control = Observer	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Combined ambient data = actual Control status: Observer
11	1.1	CTL	idle	switch = off-auto battery = on EISI-EIS: Control = Manual	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Combined ambient data = actual Control status: Manual
12	1.1	CTL	idle	switch = off-auto battery = on EISI-EIS: Control = Observer & Manual	Idle read	rpm : actual egt: actual throttle feedback: actual

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : Combined ambient data = actual Control status: Observer & Manual
13	1.2	CTL	Idle read	switch = run error = TRUE (Test Harness: Errors = switch channel)	Idle error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on engine warning panel: switch channel system status panel : actual data
14	1.2	CTL	Idle read	switch = run error = TRUE (Test Harness: Errors = throttle channel)	Idle error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on engine warning panel: throttle channel system status panel : actual data
15	1.2	CTL	Idle read	switch = run error = TRUE (Test Harness: Errors = supply voltage)	Idle error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on engine warning panel: supply voltage system status panel : actual data
16	1.3	CTL	Idle read	switch = run battery = on error = FALSE	Eng run	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: blank system status panel : actual data
17	1.10	CTL	Idle read	battery = off	idle	rpm : zero egt: default

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						throttle feedback: zero vsup : zero vout: zero engine status panel : blank engine warning panel: blank system status panel : actual data ambient data = actual
18	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = Rpm low)	eng run error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: Rpm low system status panel : actual data
19	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = switch channel)	eng run error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: switch channel system status panel : actual data
20	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = throttle channel)	eng run error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: throttle channel system status panel : actual data
21	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = egt high)	eng run error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: egt high system status panel : actual data
22	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = rpm high)	eng run error	rpm : actual egt: actual

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: rpm high system status panel : actual data
23	1.4	CTL	eng run	error = TRUE(Test Harness: Errors = ecu supply)	eng run error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch on – idle calibrated engine warning panel: ecu supply system status panel : actual data
24	1.5	CTL	eng run	battery = off	idle	rpm : zero egt: default throttle feedback: zero vsup : zero vout: zero engine status panel : blank engine warning panel: blank system status panel : actual data ambient data = actual
25	1.6	CTL	Idle error	error = TRUE (Test Harness: Errors= any) battery = off	idle	rpm : zero egt: default throttle feedback: zero vsup : zero vout: zero engine status panel : blank engine warning panel: blank system status panel : actual data ambient data = actual
26	1.7	CTL	eng run error	error = TRUE (Test Harness: Errors= any) battery = off	idle	rpm : zero egt: default throttle feedback: zero vsup : zero vout: zero engine status panel : blank engine warning panel: blank system status panel : actual data ambient data = actual

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
27	1.8	CTL	eng run	switch = off-auto	Idle read	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) - start clear engine warning panel: blank system status panel : actual data ambient data = actual
28	1.9	CTL	eng run error	error = TRUE (Test Harness: Errors= any) switch = off-auto	Idle error	rpm : actual egt: actual throttle feedback: actual vsup : actual vout: actual engine status panel : switch off (auto) engine warning panel: actual data system status panel : actual data ambient data = actual

Table D-3 - Test cases derived from SM2

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
1	2.1	LTL	Idle rpm	throttle=100%	max rpm	Setting: remains idle rpm
2	2.1	LTL	Idle rpm	throttle=20% throttle=40% throttle=60% throttle=80% throttle=100%	setting 20% setting 40% setting 60% setting 80% maxrpm	Throttle feedback ≈{20,40,60,80,100} egt ≈ {420,440,465,490,510} rpm ≈ {60x103, 75x103,95x103, 105x103, 110x103} fuel flow = proportional increase vout = proportional increase
3	2.2	CTL	max rpm	throttle = -100%	Idle rpm	Throttle feedback ≈ 0 egt ≈ 395 rpm ≈ 45x103 fuel flow ≈ idle value vout ≈ idle value
4	2.3	CTL	setting 20%	throttle = -20%	Idle rpm	Throttle feedback ≈ 0 egt ≈ 395 rpm ≈ 45x103 fuel flow ≈ idle value

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						vout ≈ idle value
5	2.4	CTL	setting 40%	throttle = -40%	Idle rpm	Throttle feedback ≈ 0 egt ≈ 395 rpm ≈ 45x103 fuel flow ≈ idle value vout ≈ idle value
6	2.5	CTL	setting 60%	throttle = -60%	Idle rpm	Throttle feedback ≈ 0 egt ≈ 395 rpm ≈ 45x103 fuel flow ≈ idle value vout ≈ idle value
7	2.6	CTL	setting 80%	throttle = -80%	Idle rpm	Throttle feedback ≈ 0 egt ≈ 395 rpm ≈ 45x103 fuel flow ≈ idle value vout ≈ idle value
8	2.7	CTL	setting 80%	throttle = -40%	setting 40%	Throttle feedback ≈ 40 egt ≈ 440 rpm ≈ 75x103 fuel flow ≈ 40% setting value vout ≈ 40% setting value
9	2.8	CTL	setting 60%	throttle = 20%	setting 80%	Throttle feedback ≈ 80 egt ≈ 490 rpm ≈ 105x103 fuel flow ≈ 80% setting value vout ≈ 80% setting value
10	2.9	CTL	setting 40%	throttle = 80%	setting 40%	Throttle feedback ≈ 40 egt ≈ 440 rpm ≈ 75x103 fuel flow ≈ 40% setting value vout ≈ 40% setting value

Table D-4 - Test cases derived from SM3

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
1	3.1	CTL	running	switch = on mains loss = FALSE (Test Harness: Mains: ON)	running	Caution panel: No Mains Loss indication
2	3.2	CTL	running	switch = on mains loss = TRUE (Test Harness: Mains: OFF or Actual Program: unplug AC Adaptor)	mains standby	Caution panel: Mains Loss indication
3	3.3	CTL	running	switch = off mains loss = FALSE (Test Harness: Mains: ON)	rapid shut	Test Harness: switch = off, engine shutting down Engine status panel: switch = off Caution panel: No Mains Loss indication
4	3.4	CTL	running	switch = auto mains loss = FALSE (Test Harness: Mains: ON)	auto shut	Test Harness: switch = auto, engine shutting down Engine status panel: switch = auto-stop Caution panel: No Mains Loss indication
5	3.5	CTL	mains standby	switch = on mains loss = FALSE (Test Harness: Mains: ON)	running	Caution panel: No Mains Loss indication
6	3.6	CTL	mains standby	switch = off mains loss = TRUE (Test Harness: Mains: OFF or Actual Program: unplug AC Adaptor)	rapid shut	Test Harness: switch = off, engine shutting down Engine status panel: switch = off Caution panel: Mains Loss indication
7	3.7	CTL	mains standby (with time in mains standby t <180 sec)	switch = auto mains loss = TRUE (Test Harness: Mains: OFF or Actual Program: unplug AC Adaptor) t<180 sec	auto shut	Test Harness: switch = auto, engine shutting down Engine status panel: switch = auto-stop Caution panel: Mains Loss indication
8	3.8	CTL	mains standby (with time in mains standby t ≥180 sec)	switch = on mains loss = TRUE (Test Harness: Mains: OFF or Actual Program: unplug AC Adaptor) t ≥180 sec	auto shut	Test Harness: switch = auto, engine shutting down Engine status panel: switch = auto-stop Caution panel: Mains Loss indication

Table D-5 - Test cases derived from SM4

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
1	CTL	4.1	running	throttle = FALSE time from throttle movement (tm) <5 input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
2	CTL	4.2	running	throttle = FALSE time from throttle movement (tm) ≥5 input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	monitor	Caution panel: Relative limits indication Ambient Pressure = actual value Ambient Temperature = actual value
3	CTL	4.4	running	throttle = FALSE time from throttle movement (tm) <5 input = absind (Test harness: Exceedance: submit RPM ≥ absolute limit)	shutdown	Caution panel: Absolute RPM limit indication Test Harness: switch =auto-stop Program shutdown Ambient Pressure = actual value Ambient Temperature = actual value
4	CTL	4.4	running	throttle = FALSE time from throttle movement (tm) <5 input = absind (Test harness: Exceedance: submit EGT ≥ absolute limit)	shutdown	Caution panel: Absolute EGT limit indication Test Harness: switch =auto-stop Program shutdown Ambient Pressure = actual value Ambient Temperature = actual value
5	CTL	4.5	monitor	throttle = FALSE input = relind (Test harness: Exceedance: submit RPM = current throttle setting nominal value < RPM <absolute limit)	standby	Caution panel: Relative RPM limit indication Ambient Pressure = actual value Ambient Temperature = actual value
6	CTL	4.5	monitor	throttle = FALSE input = relind (Test harness: Exceedance: submit EGT = current throttle setting nominal value < EGT <absolute limit)	standby	Caution panel: Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value
7	CTL	4.6	monitor	throttle = FALSE input = absind (Test harness: Exceedance: submit RPM ≥ absolute limit)	shutdown	Caution panel: Absolute RPM limit indication Ambient Pressure = actual value Ambient Temperature = actual value Test Harness: switch =auto-stop Program shutdown
8	CTL	4.6	monitor	throttle = FALSE input = absind (Test harness: Exceedance: submit EGT ≥ absolute limit)	shutdown	Caution panel: Absolute EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value Test Harness: switch =auto-stop Program shutdown
9	CTL	4.7	monitor	throttle = TRUE input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
10	CTL	4.8	standby (no	throttle = FALSE	shutdown	Caution panel: Relative RPM limit indication

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
			absolute exceedance)	input = relind (Test harness: Exceedance: submit RPM = current throttle setting nominal value < RPM <absolute limit) time in stby (ts) ≥ 180 sec		Ambient Pressure = actual value Ambient Temperature = actual value After 3 minutes: Test Harness: switch =auto-stop, Program shutdown
11	CTL	4.8	standby (no absolute exceedance)	throttle = FALSE input = relind (Test harness: Exceedance: submit EGT = current throttle setting nominal value < EGT <absolute limit) time in stby (ts) ≥ 180 sec	shutdown	Caution panel: Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value After 3 minutes: Test Harness: switch =auto-stop, Program shutdown
12	CTL	4.8	standby (no absolute exceedance)	throttle = FALSE input = relind (Test harness: Exceedance: submit EGT & RPM = current throttle setting nominal value < EGT & RPM <absolute limit) time in stby (ts) ≥ 180 sec	shutdown	Caution panel: Relative EGT limit indication & Relative RPM indication Ambient Pressure = actual value Ambient Temperature = actual value After 3 minutes: Test Harness: switch =auto-stop, Program shutdown
13	CTL	4.9	standby (absolute exceedance present)	Throttle = FALSE Input = relind (Test harness: exceedance: submit RPM = current throttle setting nominal value < RPM <absolute limit) Time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit RPM ≥ absolute limit)	shutdown	Caution panel: Relative RPM limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute RPM limit indication Test Harness: switch =auto-stop Program shutdown
14	CTL	4.9	standby (absolute exceedance present)	throttle = FALSE input = relind (Test harness: Exceedance: submit RPM = current throttle setting nominal value < RPM <absolute limit) time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit EGT ≥ absolute limit)	shutdown	Caution panel: Relative RPM limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute EGT limit indication Test Harness: switch =auto-stop Program shutdown
15	CTL	4.9	standby (absolute exceedance present)	throttle = FALSE input = relind (Test harness: Exceedance: submit EGT = current throttle setting nominal value < EGT <absolute limit) time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit RPM ≥ absolute limit)	shutdown	Caution panel: Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute RPM limit indication Test Harness: switch =auto-stop Program shutdown
16	CTL	4.9	standby (absolute exceedance present)	throttle = FALSE input = relind (Test harness: Exceedance: submit EGT = current throttle setting nominal value < EGT <absolute limit) time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit EGT ≥ absolute limit)	shutdown	Caution panel: Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute EGT limit indication

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						Test Harness: switch =auto-stop Program shutdown
17	CTL	4.9	standby (absolute exceedance present)	throttle = FALSE input = relind (Test harness: Exceedance: submit RPM & EGT = current throttle setting nominal value < RPM & EGT <absolute limit) time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit RPM ≥ absolute limit)	shutdown	Caution panel: Relative RPM limit indication & Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute RPM limit indication Test Harness: switch =auto-stop Program shutdown
18	CTL	4.9	standby (absolute exceedance present)	throttle = FALSE input = relind (Test harness: Exceedance: submit RPM & EGT = current throttle setting nominal value < RPM & EGT <absolute limit) time in stby (ts) < 180 sec input = absind (Test harness: Exceedance: submit EGT ≥ absolute limit)	shutdown	Caution panel: Relative RPM limit indication & Relative EGT limit indication Ambient Pressure = actual value Ambient Temperature = actual value Prior to 3 minutes: Absolute EGT limit indication Test Harness: switch =auto-stop Program shutdown
19	CTL	4.10	Standby (egt relative indication)	throttle = FALSE input = normal (Test Harness: Exceedance: clear) input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
20	CTL	4.10	Standby (rpm relative indication)	throttle = FALSE input = normal (Test Harness: Exceedance: clear) input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
21	CTL	4.10	standby (rpm & egt relative indications)	throttle = FALSE input = normal (Test Harness: Exceedance: clear) input = relind (Test harness: Exceedance: submit RPM or EGT = current throttle setting nominal value < RPM or EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
22	CTL	4.11	standby (persistent rpm relative indication)	time in stby (ts) < 180 sec throttle =TRUE input = relind (Test harness: Exceedance: Do not clear RPM = current throttle setting nominal value < RPM <absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
23	CTL	4.11	standby (persistent egt relative indication)	time in stby (ts) < 180 sec throttle =TRUE input = relind (Test harness: Exceedance: Do not clear EGT = current throttle setting nominal value < EGT<absolute limit)	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value
24	CTL	4.11	standby (persistent rpm & egt	time in stby (ts) < 180 sec throttle =TRUE input = relind (Test harness: Exceedance: Do not clear RPM & EGT =	running	Caution panel: No indications Ambient Pressure = actual value Ambient Temperature = actual value

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
			relative indications)	current throttle setting nominal value < RPM or EGT<absolute limit)		

Table D-6 - Test cases derived from SM5

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
1	5.1	CTL	run normal	emergency stop = TRUE	auto stop	- switch = auto stop - engine shutting down
2	5.2	CTL	run normal	switch signal = lost: disconnect port 1 of NI 9263 module	signal loss (switch signal loss present)	- "take manual control" warning on at observer's panel - switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
3	5.2	CTL	run normal	throttle signal = lost: disconnect port 3 of NI 9263 module	signal loss (throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
4	5.2	CTL	run normal	throttle signal = lost: disconnect ports 3 & 4 of NI 91263 module	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
5	5.2	CTL	run normal	throttle signal = lost: disconnect or unplug NI DAQ 9174 unit	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 1 & 2 cease to illuminate on EDT warning panel
6	5.3	CTL	run normal	observer switch = TRUE: select local override on at observer's panel	run observer	- Engine control status on browser& observer's panel = "Observer". - Engine does not listen to any command from browser or manual control panel
7	5.4	CTL	run normal	Input=lost: disconnect USB port 3	Input loss	- "take manual control" warning on at observer's panel

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
						- serial input warning on at observer's panel
8	5.5	CTL	run normal	manual switch = TRUE: on Junction Box place all 5 switches from PC to MANUAL	run manual	- Engine control status on browser& observer's panel = "Manual". - Engine does not listen to any command from browser or observer's panel
9	5.6	CTL	run observer	emergency stop = TRUE	auto stop	- switch = auto stop - engine shutting down
10	5.7	CTL	run observer	switch signal = lost: disconnect port 1 of NI 9263 module	signal loss (switch signal loss present)	- "take manual control" warning on at observer's panel - switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
11	5.7	CTL	run observer	throttle signal = lost: disconnect port 3 of NI 9263 module	signal loss (throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
12	5.7	CTL	run observer	throttle signal = lost: disconnect ports 3 & 4 of NI 91263 module	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
13	5.7	CTL	run observer	throttle signal = lost: disconnect or unplug NI DAQ 9174 unit	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 1 & 2 cease to illuminate on EDT warning panel
14	5.8	CTL	run observer	observer switch = TRUE: select local override off at observer's panel	run normal	- Engine control status on browser& observer's panel = "Normal". - Engine does not listen to any command from observer's panel or manual control panel
15	5.9	CTL	run observer	Input=lost: disconnect USB port 3	Input loss	- "take manual control" warning on at observer's panel - serial input warning on at observer's panel
16	5.10	CTL	run observer	manual switch = TRUE: on Junction Box place all 5 switches from PC to MANUAL	run manual	- Engine control status on browser& observer's panel = "Manual". - Engine does not listen to any command from browser or observer's panel

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
17	5.11	CTL	signal loss	emergency stop = TRUE	auto stop	- switch = auto stop - engine shutting down
18	5.12	CTL	signal loss	manual switch = TRUE: on Junction Box place all 5 switches from PC to MANUAL	run manual	- Engine control status on browser& observer's panel = "Manual". - Engine does not listen to any command from browser or observer's panel
19	5.13	CTL	input loss	Bring to state input loss and then just monitor for time delay t=30 ms	auto stop	- switch = auto stop - engine shutting down
20	5.14	CTL	run manual	- observer switch = FALSE: select local override off at observer's panel - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	run normal	- Engine control status on browser& observer's panel = "Normal". - Engine does not listen to any command from observer's panel or manual control panel
21	5.15	CTL	run manual	- observer switch = TRUE: select local override on at observer's panel - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	run observer	- Engine control status on browser& observer's panel = "Observer". - Engine does not listen to any command from browser or manual control panel
22	5.16	CTL	run manual	- switch signal = lost: disconnect port 1 of NI 9263 module - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	signal loss (switch signal loss present)	- "take manual control" warning on at observer's panel - switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
23	5.16	CTL	run manual	- throttle signal = lost: disconnect port 3 of NI 9263 module - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	signal loss (throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
24	5.16	CTL	run manual	- throttle signal = lost: disconnect ports 3 & 4 of NI 91263 module - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 3 & 4 illuminated on EDT warning panel
25	5.16	CTL	run manual	- throttle signal = lost: disconnect or unplug NI DAQ 9174 unit - manual switch = FALSE: on Junction Box place all 5 switches from PC to MANUAL	signal loss (switch and throttle signal loss present)	- "take manual control" warning on at observer's panel - throttle & switch signal warning on at observer's panel - indication lights 1 & 2 cease to illuminate on EDT warning panel
26	5.17	CTL	run manual	manual stop = TRUE: manual control panel: set run start switch to "Auto-stop"	auto-stop	- switch = auto stop - engine shutting down

Test Case No	Spec No	Spec Type	Initial state	Parameters	Next State	Assertions
27	5.18	CTL	signal loss	Input=lost: disconnect USB port 3	Input loss	- “take manual control” warning on at observer’s panel - serial input warning on at observer’s panel

D.3 Acceptance validation tables

Each row of the table indicates a functional requirement to be tested (format: Function mode.Functional requirement.Sub requirement). Tests conducted beyond the functional requirements are noted as –additional (format: Function mode.Functional requirement.-additional-number). Each row of requirements includes the following attributes (columns): Number of requirement, state machine where this function is present, test cases applied, method of testing (with the EIS and gas turbine engaged or with the General Test Harness instead), the results of the test and finally, any recommendations.

Requirement	**Corresponding State Machine**	**Test Case No**	**Method (AP = Actual Program) (TH = Tests Harness)**	**Results**	**Recommendations**
0.1.1	Sm.0	0.2	TH - AP	OK	
0.1.2	Sm.0	0.1	TH - AP	OK	
0.1.3	Sm.0	0.3	TH - AP	OK	
0.1.4	Sm.0	0.3	TH - AP	OK	
0.1.5	Sm.0	0.8 – 0.9	TH - AP	OK	
0.1.6	Sm.0	0.6 – 0.7	TH – AP	OK	
0.1.7	Sm.0	0.6 – 0.8	TH – AP	OK	
0.1.8	Sm.0	0.4	TH – AP	OK	
0.1- additional-1	Sm.0	0.5	TH – AP	OK	
0.1-additional-2	Sm.0	0.10	TH – AP	OK	
0.1-additional-3	Sm.0	0.11	TH – AP	OK	
0.1-additional-2	Sm.0	0.12	TH – AP	OK	
0.1-additional-3	Sm.0	0.13	TH – AP	OK	
0.2.1	Sm.0	0.3	TH – AP	Flowplayer error 201: Stream not found	Renew Wowza Media Server 3.6.2 License
1.1.1	Sm.1	1.16	TH – AP	OK	
1.1.2	Sm.1	1.16	TH – AP	OK	
1.1.3	Sm.1	1.16	TH – AP	OK	
1.1.4	Sm.1	1.16	TH – AP	OK	
1.1.5	Sm.1	1.16	TH – AP	OK	
1.1.6	Sm.1	1.16	TH – AP	OK	
1.1.7	Sm.1	1.16	TH – AP	OK	
1.1.8	Sm.1	1.16	TH – AP	OK	
1.1.9	Sm.1	1.16	TH – AP	OK	
1.1.10	-	-	TH – AP	-	Suspended

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
1.1.11	Sm.1	1.16	TH – AP	OK	
1.1.12	Sm.1	1.16	TH – AP	OK	
1.1.13	Sm.1	1.16	TH – AP	OK	
1.1.14	Sm.1	1.16	TH – AP	OK	
1.1.15	Sm.1	1.16	TH – AP	OK	
1.1.16	-	-		-	Suspended
1.1.17	Sm.1	1.16	TH – AP	OK	
1.1.18	Sm.1	1.16	TH – AP	OK	
1.1.19	Sm.1	1.16	TH – AP	OK	
1.1.20	Sm.1	1.16	TH – AP	OK	
1.1.21	Sm.1	1.16	TH – AP	OK	
2.1.1	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.2	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.3	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.4	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.5	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.6	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.7	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.8	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.9	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.10	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.11	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.12	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.13	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.14	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
2.1.15	Sm.1	1.13 – 1.14 – 1.15	TH	OK	
1.1- additional-1	Sm.1	1.2	TH	OK	
1.1- additional-2	Sm.1	1.3	TH	OK	
1.1- additional-3	Sm.1	1.4	TH	OK	
1.1- additional-4	Sm.1	1.5	TH	OK	
1.1- additional-5	Sm.1	1.6	TH	OK	
1.1- additional-6	Sm.1	1.7	TH	OK	
1.1- additional-7	Sm.1	1.8	TH - AP	OK	
1.1- additional-8	Sm.1	1.9	TH - AP	OK	
1.1- additional-9	Sm.1	1.10	TH - AP	OK	
1.1- additional-10	Sm.1	1.11	TH - AP	OK	

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
1.1- additional-11	Sm.1	1.12	TH - AP	OK	
1.1- additional-12	Sm.1	1.17	TH - AP	OK	
1.1- additional-13	Sm.1	1.24	TH - AP	OK	
1.1- additional-14	Sm.1	1.26	TH	OK	
1.1- additional-15	Sm.1	1.27	TH-AP	OK	
1.1- additional-16	Sm.1	1.28	TH	OK	
2.1- additional-1	Sm.1	1.18	TH	OK	
2.1- additional-2	Sm.1	1.19	TH	OK	
2.1- additional-3	Sm.1	1.20	TH	OK	
2.1- additional-4	Sm.1	1.21	TH	OK	
2.1- additional-5	Sm.1	1.22	TH	OK	
2.1- additional-6	Sm.1	1.23	TH	OK	
2.1- additional-6	Sm.1	1.25	TH	OK	
3.1.1	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.2	Sm.2	2.1 – 2.10	TH-AP	OK	
3.1.3	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.4	Sm.2	2.1 – 2.10	TH-AP	OK	
3.1.5	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.6	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.7	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.8	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.9	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.10	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.11	Sm.2	-	-	-	Suspended
3.1.12	Sm.2	-	-	-	Suspended
3.1.13	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.14	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.15	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.16	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.17	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.18	Sm.2	2.2 – 2.9	-	-	Suspended
3.1.19	Sm.2	2.2 – 2.9	-	-	Suspended
3.1.20	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.21	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.22	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.23	Sm.2	2.2 – 2.9	TH-AP	OK	

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
3.1.24	Sm.2	2.2 – 2.9	TH-AP	OK	
3.1.25	Sm.2	2.1 – 2.10	TH-AP	OK	
3.2.1	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.2	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.3	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.4	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.5	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.6	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.7	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.8	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.9	Sm.2	-	-	-	Suspended
3.2.10	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.11	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.12	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.13	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.14	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.15	Sm.2	-	-	-	Suspended
3.2.16	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.17	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.18	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.19	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.2.20	Sm.2	2.3 – 2.4 – 2.5 – 2.6 – 2.7 – 2.8	TH-AP	OK	
3.3.1		-	-	-	Suspended
3.3.2		-	-	-	Suspended
3.3.3		-	-	-	Suspended
4.1.1	Sm.4	4.2	TH	OK	
4.1.2	Sm.4	4.2	TH	OK	
4.1.3	Sm.4	4.3– 4.7 – 4.13 – 4.15	TH	OK	
4.1.4	Sm.4	4.4 – 4.8 – 4.14 – 4.16	TH	OK	
4.1.5	Sm.4	4.3– 4.7 – 4.13 – 4.15	TH	OK	
4.1.6	Sm.4	4.4 – 4.8 – 4.14 – 4.16	TH	OK	
4.1.7	Sm.4	4.3– 4.4 - 4.7 – 4.8 – 4.13 – 4.14 - 4.15 – 4.16	TH	OK	
4.1.8	Sm.4	4.4 – 4.8 – 4.14 – 4.16	TH	OK	
4.1.9	Sm.4	4.3– 4.7 – 4.13 – 4.15	TH	OK	
4.2.1	Sm.4	4.1	TH - AP	OK	

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
4.2.2	Sm.4	4.1	TH - AP	OK	
4.2.3	Sm.4	4.2	TH	OK	
4.2.4	Sm.4	4.2	TH	OK	
4.2.5	Sm.4	4.2	TH	OK	
4.2.6	Sm.4	4.2	TH	OK	
4.2.7	Sm.4	4.2	TH	OK	
4.2.8	Sm.4	4.2	TH	OK	
4.2.9	Sm.4	4.5	TH	OK	
4.2.10	Sm.4	4.6	TH	OK	
4.2.11	Sm.4	4.10 – 4.11 – 4.12	TH	OK	
4.2.12	Sm.4	4.5	TH	OK	
4.2.13	Sm.4	4.6	TH	OK	
4.2.14	Sm.4	4.1	TH	OK	
4.2.15	Sm.4	4.1	TH	OK	
4.2.16	Sm.4	4.5	TH	OK	
4.2.17	Sm.4	4.6	TH	OK	
4.2- additional-1	Sm.4	4.9	TH	OK	
4.2- additional-2	Sm.4	4.17	TH	OK	
4.2- additional-3	Sm.4	4.18	TH	OK	
4.2- additional-4	Sm.4	4.19	TH	OK	
4.2- additional-5	Sm.4	4.20	TH	OK	
4.2- additional-6	Sm.4	4.21	TH	OK	
4.2- additional-7	Sm.4	4.22	TH	OK	
4.2- additional-8	Sm.4	4.23	TH	OK	
4.2- additional-9	Sm.4	4.24	TH	OK	
4.3.1	Sm.3	3.1 – 3.2	TH - AP	OK	
4.3.2	Sm.5	5.2 – 5.3 – 5.4 – 5.10 – 5.11 – 5.12	AP	OK	
4.3.3	Sm.5	5.5 – 5.13	AP	OK	
4.3.4	Sm.5	5.7 – 5.9 - 5.18	AP	OK	
4.3.5	Sm.5	5.6	TH- AP	OK	
4.3.6	Sm.3	3.1 – 3.2	TH - AP	OK	
4.3.7	Sm.3	3.8	TH - AP	OK	
4.3.8	Sm.3	3.1 – 3.2	TH - AP	OK	
4.3.9	Sm.5	5.6	TH- AP	OK	
4.3.10	Sm.3	3.1 – 3.2	TH - AP	OK	

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
4.3.11	Sm.5	5.6	TH- AP	OK	
4.3- additional-1	Sm.3	3.5	TH	OK	
4.3- additional-2	Sm.3	3.6	TH	OK	
4.3- additional-3	Sm.3	3.7	TH	OK	
4.3- additional-4	Sm.5	5.27	TH- AP	OK	
5.1.1	Sm.3	3.4	TH- AP	OK	
5.1.2	Sm.3	3.3	TH- AP	OK	
5.1.3	Sm.3	3.4	TH- AP	OK	
5.1.4	Sm.3	3.3	TH- AP	OK	
5.1.5	Sm.3	3.4	TH- AP	OK	
5.1.6	Sm.3	3.3	TH- AP	OK	
5.1.7	-	-	-	-	Suspended
5.1.8	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.9	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.10	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.11	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.12	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.13	-	-	-	-	Suspended
5.1.14	-	-	-	-	Suspended
5.1.15	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.16	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.17	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.18	Sm.3	3.3 – 3.4	TH- AP	OK	
5.1.19	Sm.3	3.3 – 3.4	TH- AP	OK	
5.2.1	Sm.5	5.1	TH- AP	OK	
5.2.2	Sm.5	5.7 - 5.15 – 5.18 – 5.19	AP	OK	
5.2.3	Sm.5	5.26	AP	OK	
5.2.4	-	-	-	-	Suspended
5.2.5	Sm.5	5.8 – 5.10 – 5.18	TH-AP	OK	
5.2.6	Sm.5	5.1	TH- AP	OK	
5.2.7	Sm.5	5.1	TH- AP	OK	
5.2.8	Sm.5	5.1	TH- AP	OK	
5.2.9	Sm.5	5.1	TH- AP	OK	
5.2.10	Sm.5	5.1	TH- AP	OK	
5.2.11	-	-	-	-	Suspended
5.2.12	-	-	-	-	Suspended

Requirement	Corresponding State Machine	Test Case No	Method (AP = Actual Program) (TH = Tests Harness)	Results	Recommendations
5.2.13	Sm.5	5.8 – 5.10 – 5.18	TH-AP	OK	
5.2.14	Sm.5	5.1	TH- AP	OK	
5.2.15	Sm.5	5.1	TH- AP	OK	
5.2.16	Sm.5	5.1	TH- AP	OK	
5.2.17	Sm.5	5.1	TH- AP	OK	
5.2.18	Sm.5	5.1	TH- AP	OK	
5.2- additional-1	Sm.5	5.14	TH- AP	OK	
5.2- additional-2	Sm.5	5.16	TH- AP	OK	
5.2- additional-3	Sm.5	5.17	AP	OK	
5.2- additional-4	Sm.5	5.20	TH- AP	OK	
5.2- additional-5	Sm.5	5.21	TH- AP	OK	
5.2- additional-6	Sm.5	5.22	AP	OK	
5.2- additional-7	Sm.5	5.23	AP	OK	
5.2- additional-8	Sm.5	5.24	AP	OK	
5.2- additional-9	Sm.5	5.25	AP	OK	

Appendix E Software Components Documentation

E.1 LabVIEW Modules – EIS.vi

E.1.1 SwitchSignal function

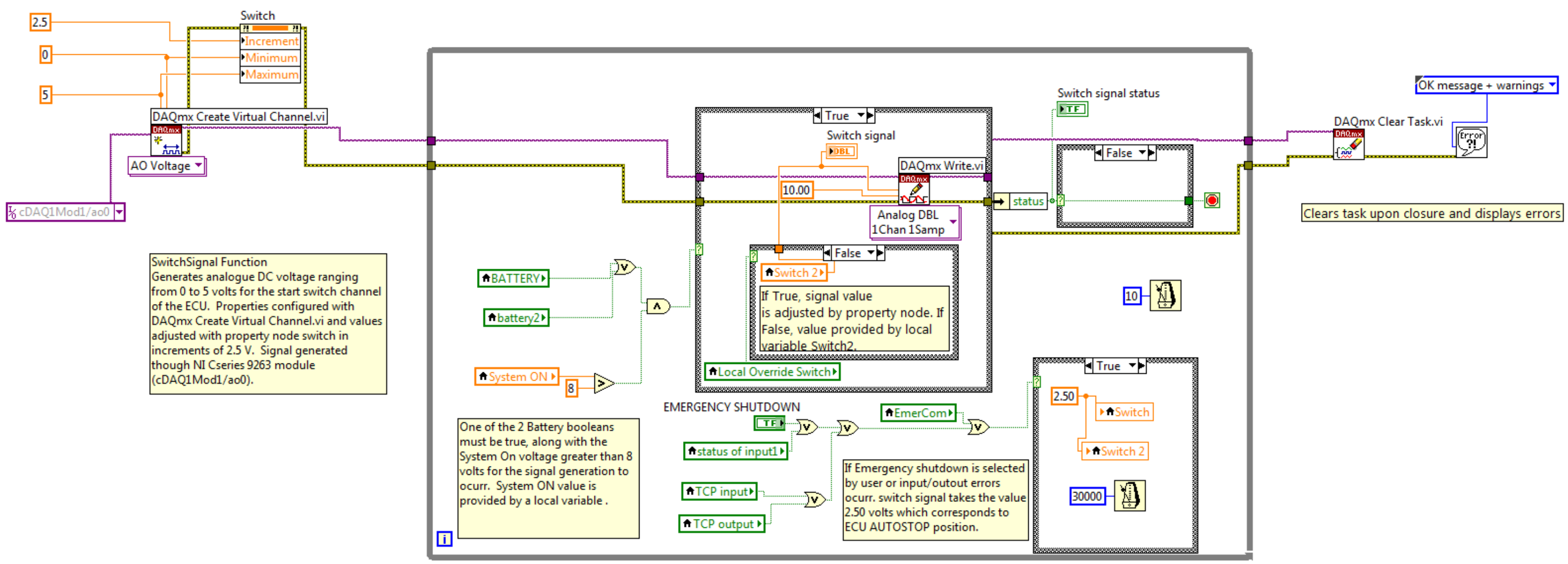


Figure E-1 – SwitchSignal function block diagram

E.1.2 ThrottleSignal function

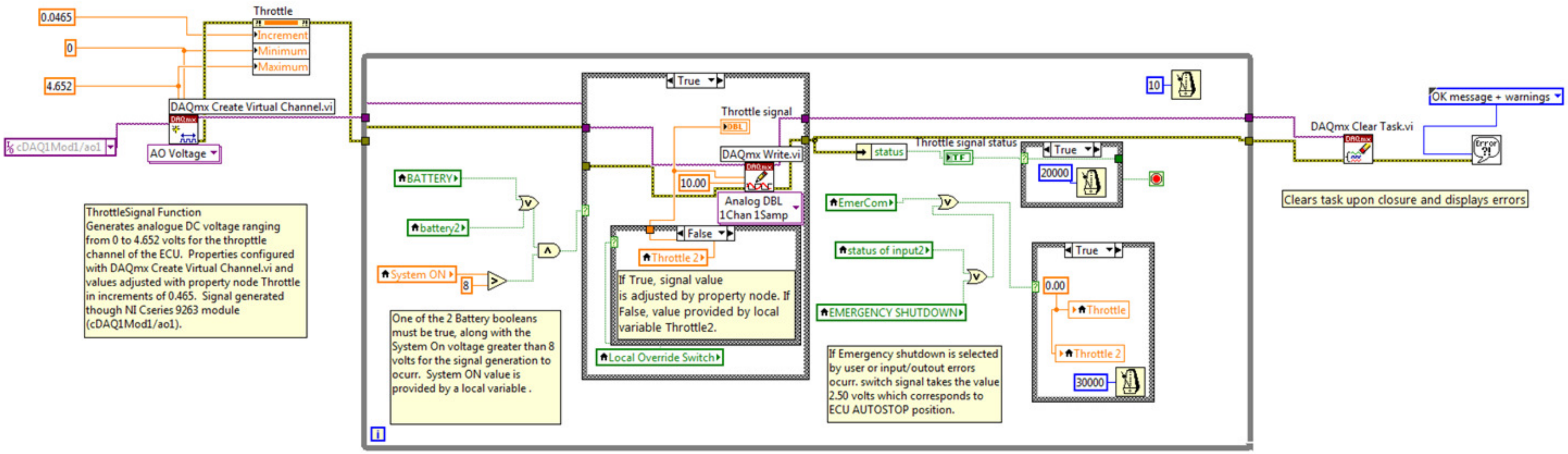


Figure E-2 – ThrottleSignal function block diagram

E.1.3 ECUSerial function

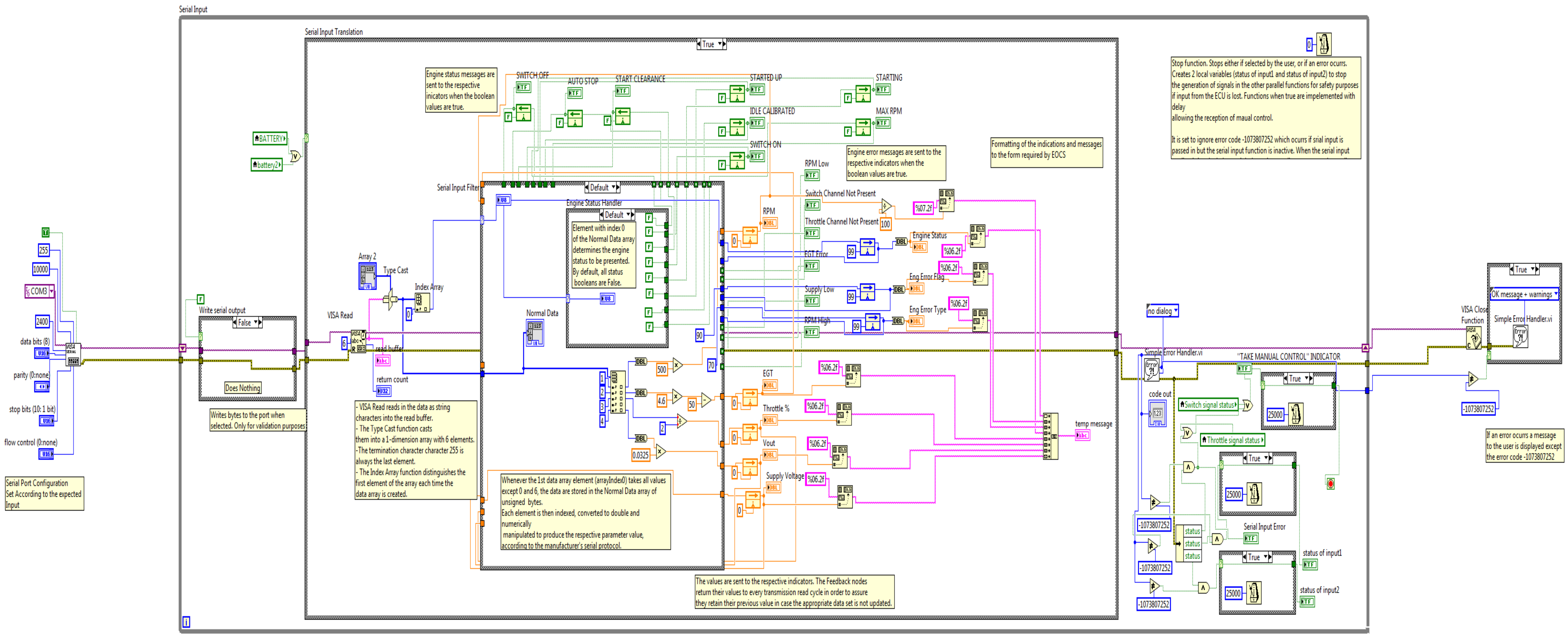


Figure E-3 – ECUSerial function block diagram

E.1.4 Fuel flow acquisition - Utilities signal functions

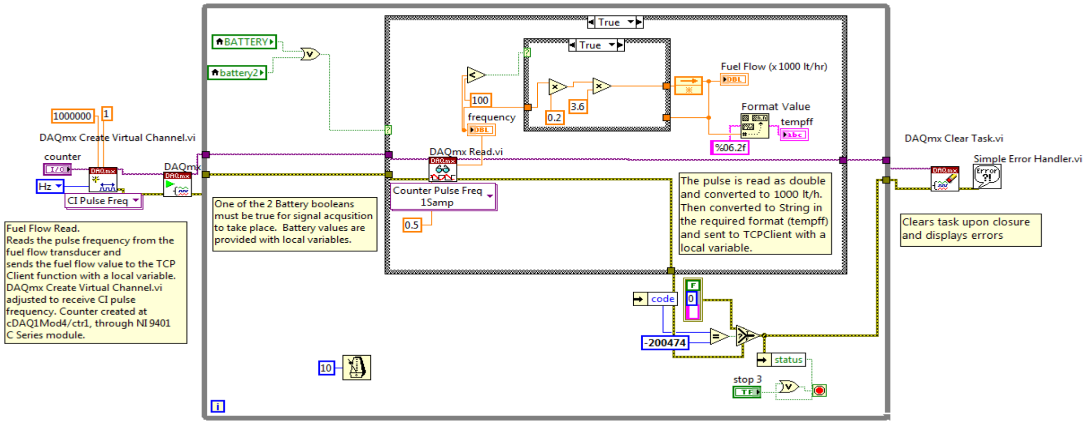


Figure E-4 – FuelFlowRead function

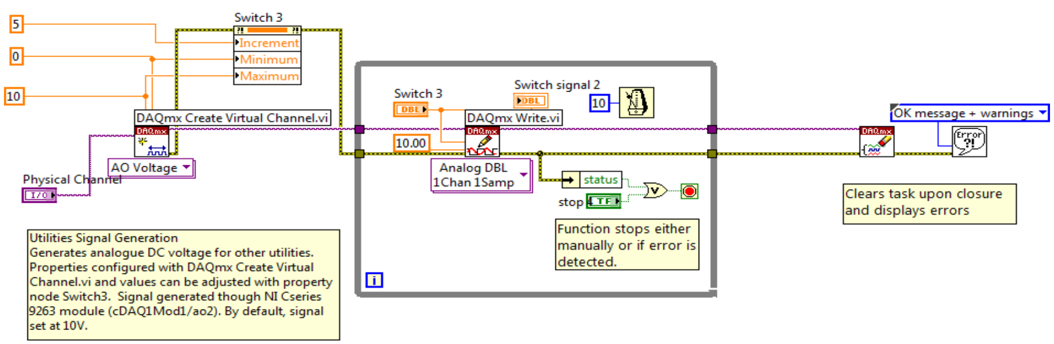


Figure E-5 UtilitiesSignalGeneration function block diagram

E.1.5 UtilitiesSignalAcquisition function

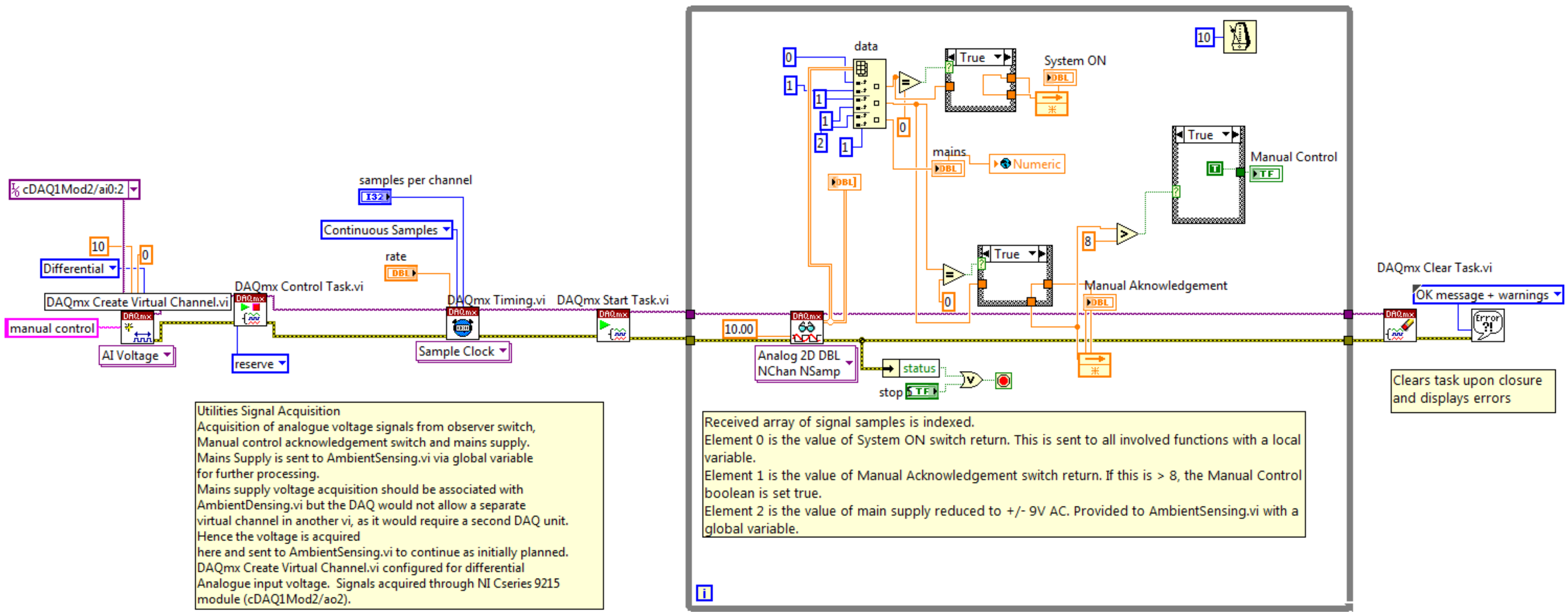


Figure E-6 – UtilitiesSignalAcquisition function block diagram

E.1.6 EDT warning signal generation – Remote emergency shutdown function

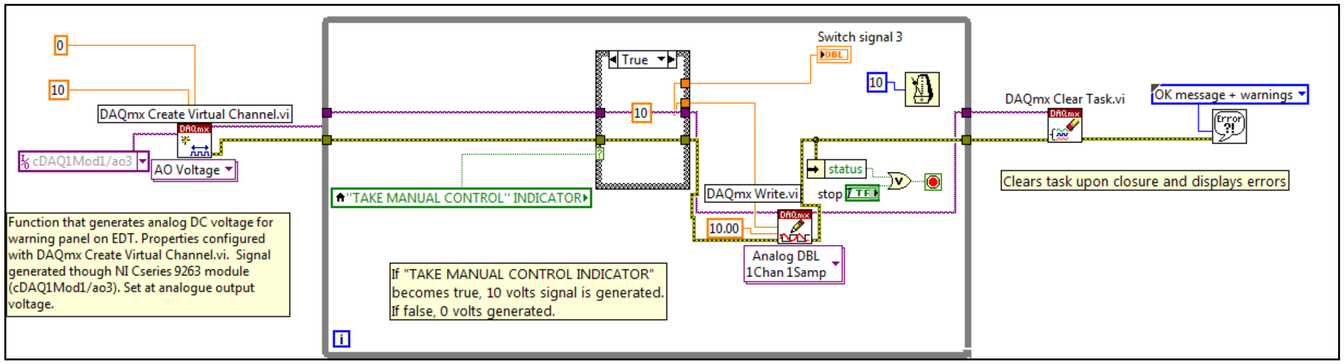


Figure E-7 – EDTWarning function block diagram

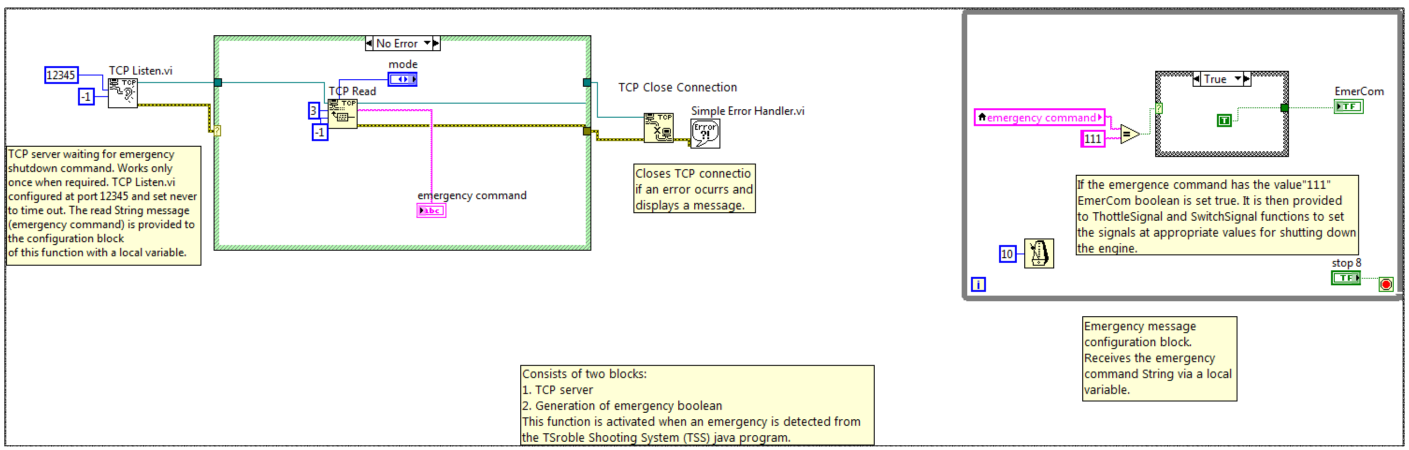


Figure E-8 RemoteEmergencyShutdown function block diagram

E.1.7 TCPClient and TCPServer functions

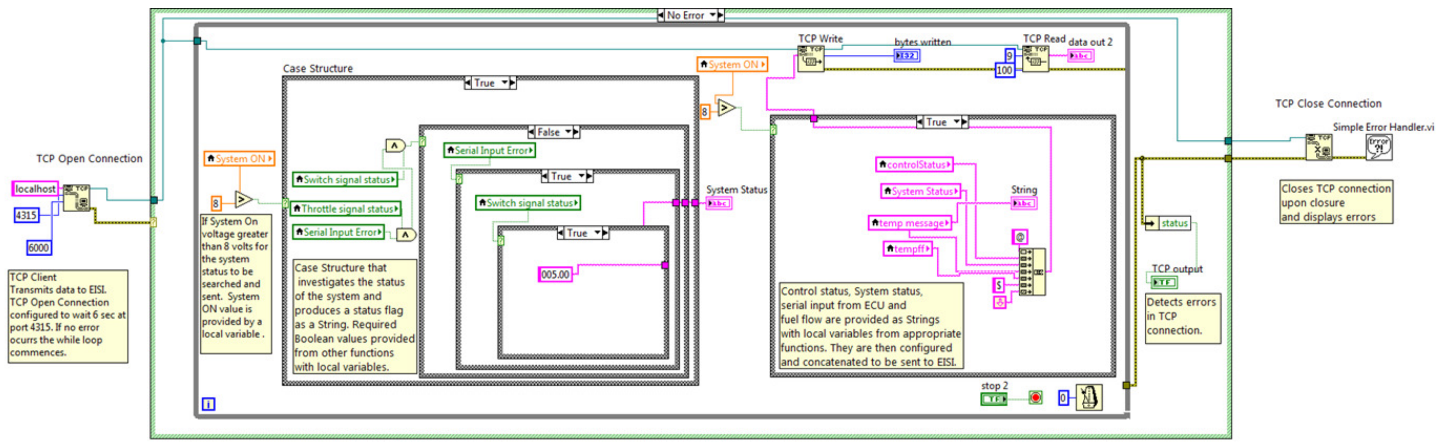


Figure E- 9 – TCPClient function block diagram

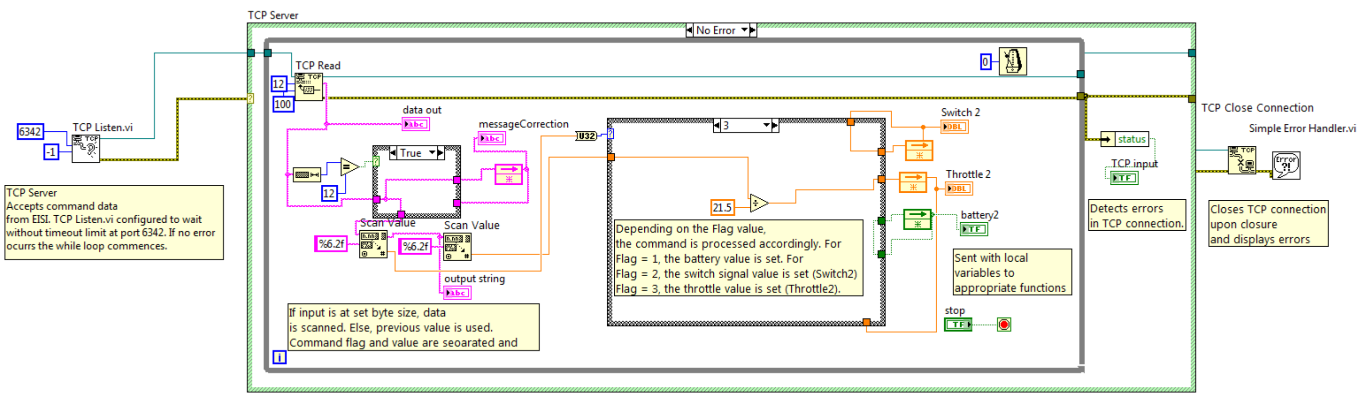


Figure E-10 - TCPServer function block diagram

E.2 AmbientSensing.vi

E.2.1 AmbientTemperature function

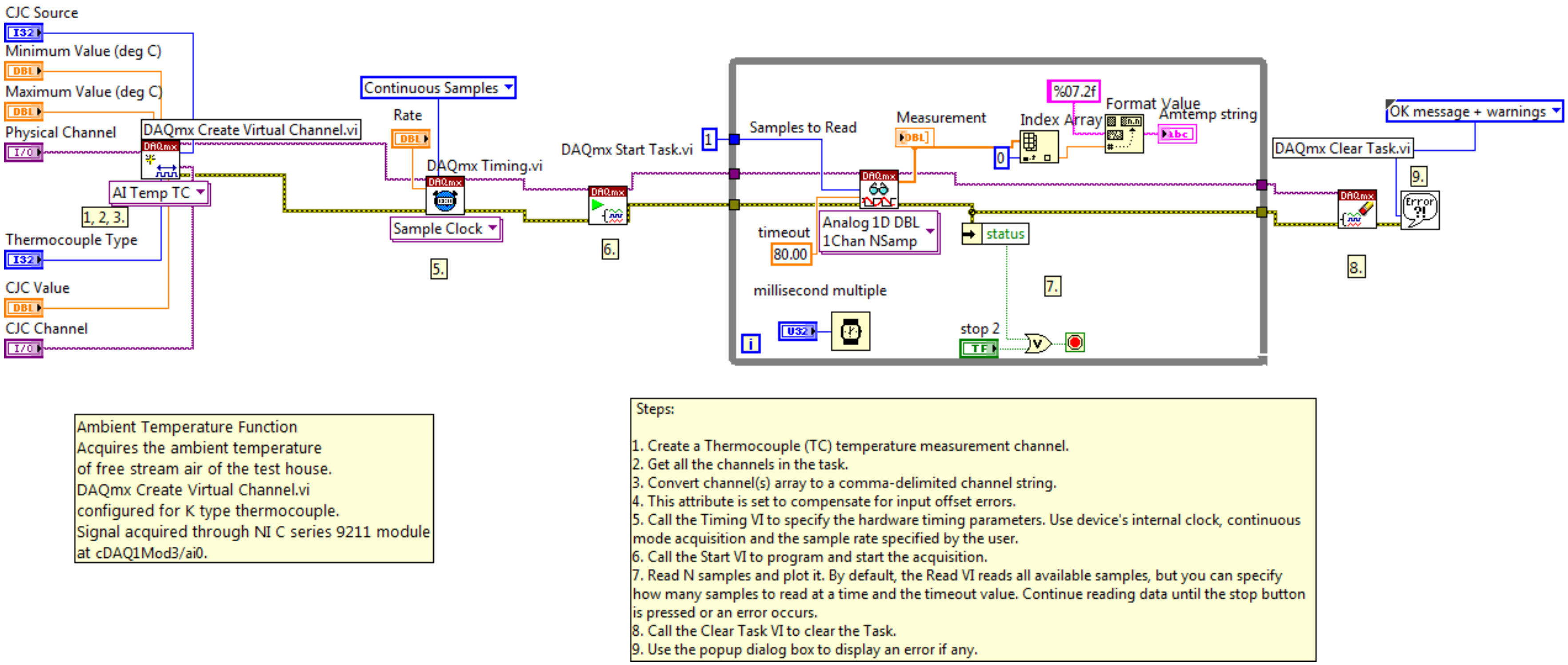


Figure E-11 – AmbientTemperature function block diagram

E.2.2 Mains Supply Interface System (MSIS)

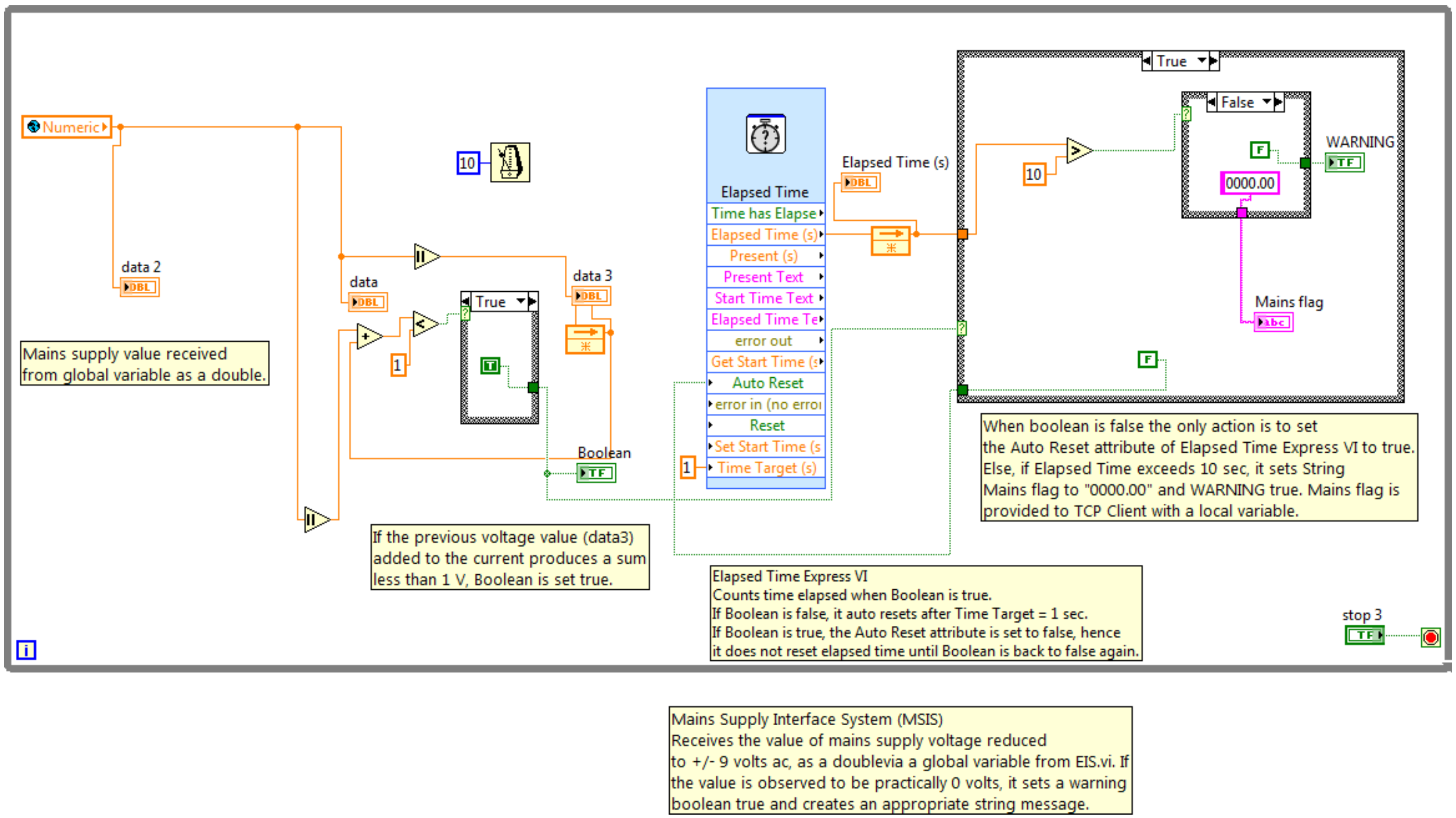


Figure E-12 – MainsSupplyInterfaceSystem function block diagram

E.2.3 TCPClient function

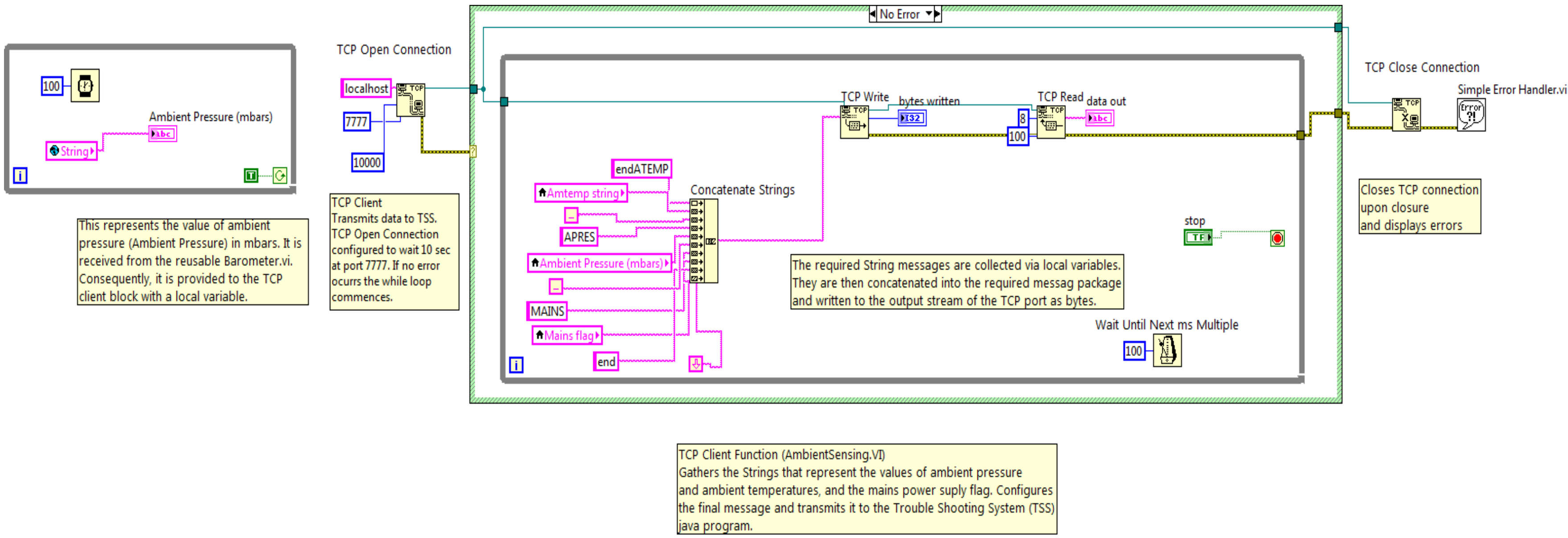


Figure E-13 – AmbientSensing.vi – TCPClient function block diagram

E.3 SignalClosure.vi

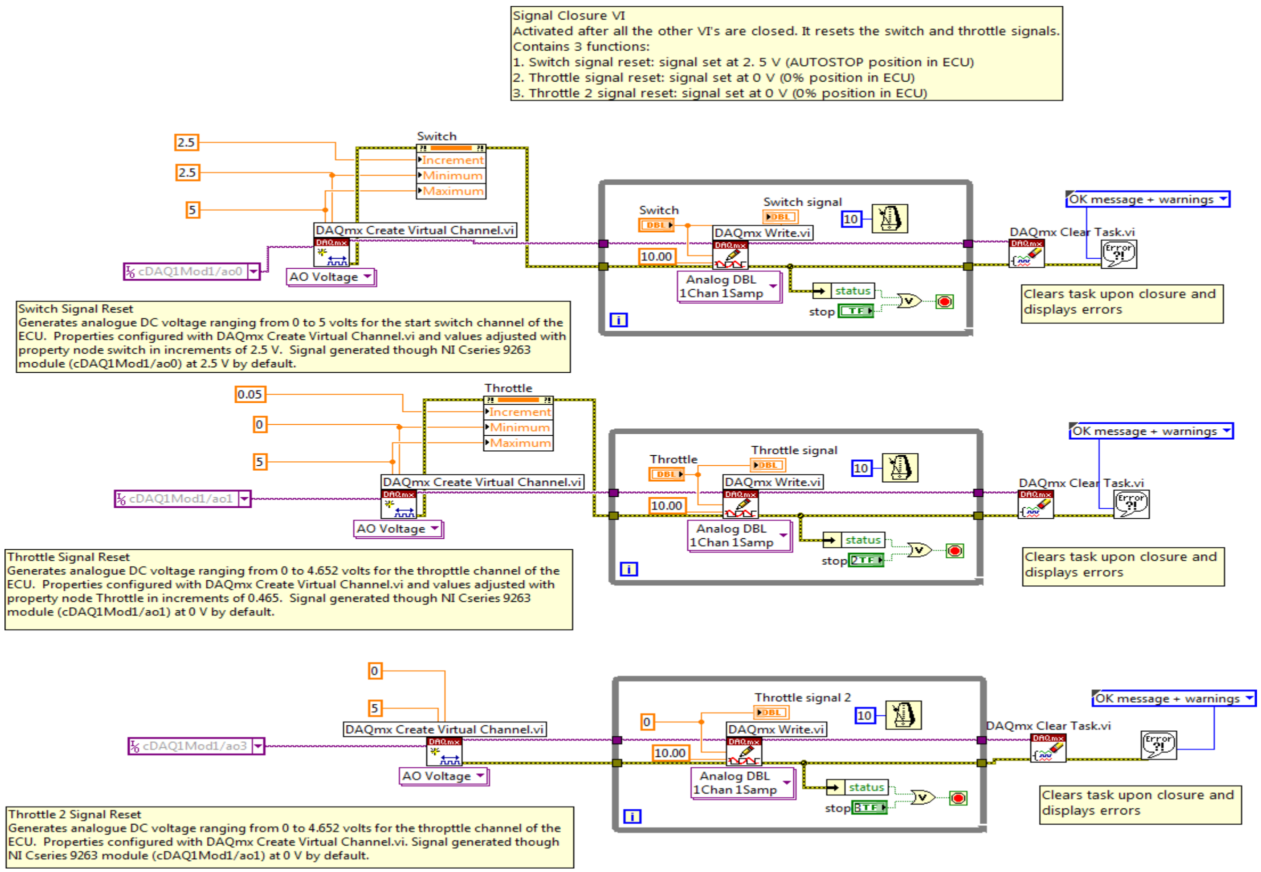


Figure E-14 – SignalClosure.vi block diagram

E.4 Java modules

Due to extensive length, only the summary of the fields are shown in this document. The details are available in the original Javadoc files of the modules.

E.4.1 Application EISI – package EISI – class AmbientIn.Java

Package EISI
Class AmbientIn
java.lang.Object
EISI.AmbientIn
All Implemented Interfaces:
java.lang.Runnable
`public class AmbientIn`
`extends java.lang.Object`
`implements java.lang.Runnable`
Runs on a dedicated Thread. Acquires ambient and peripheral data transmitted from the TSS and transfers them to the
MEOCS.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`java.net.InetAddress addr`
Private.
`java.lang.String ambientTSSInfo`
Private.
`java.lang.String hostIpAddress`
Private.
`java.io.BufferedReader inFromMEOCS`
Private.
`java.io.BufferedReader inFromTSS`
Private.
`java.io.DataOutputStream outToMEOCS`
Private.
`java.io.DataOutputStream outToTSS`
Private.
`int port`
Private.
`int port1`
Private.
`java.lang.Thread runner`
Private.
Fields
`int time`
Private.
Constructor Summary
Constructor and Description
`AmbientIn(java.lang.String hostAddress)`
Default Constructor.
`AmbientIn(java.lang.String threadname, java.lang.String hostAddress)`
Constructor to start the thread.
Method Summary
Modifier and Type Method and Description
`protected void attachShutDownHook(java.net.Socket sendMEOCS)`
Attaches a ShutDown Hook.
`void run()`
Run method.

E.4.2 Application EISI – package EISI – class CamCapture.Java

Package EISI
Class CamCapture
java.lang.Object
EISI.CamCapture
All Implemented Interfaces:
java.lang.Runnable
`public class CamCapture`
`extends java.lang.Object`
`implements java.lang.Runnable`
Instantiates a Thread that captures and transmits live video image. This class is called only in LAN use. I imports the
opencv and javacv libraries.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis

Field Summary
Modifier and Type Field and Description
`int bytes`
Private.
`FrameGrabber grabber`
Private.
`IplImage img`
Private.This is an opencv object.It stores the captured image.
`int port`
Private.
`java.lang.Thread runner`
Private.
`java.net.DatagramSocket serverSocket`
Private.
Constructor Summary
Constructor and Description
Fields
Constructors
`CamCapture()`
Default constructor.
`CamCapture(java.lang.String threadname)`
Constructor to start the camCaptureThread thread.
Method Summary
Modifier and Type Method and Description
`void attachShutDownHook()`
Shutdown hook to stop camera when program is closing.
`void run()`
Run method.

E.4.3 Application EISI – package EISI – class ClientEIS.Java

Package EISI
Class ClientEIS
java.lang.Object
EISI.ClientEIS
All Implemented Interfaces:
java.lang.Runnable
`public class ClientEIS`
`extends java.lang.Object`
`implements java.lang.Runnable`
Runs on a dedicated thread. Accepts command messages from the MEOCS and transmits them to the EIS for signal
generation. Also checks the throttle setting increase command if rapid (>20%).
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`double check`
Private.
`java.net.Socket client`
Private.
`java.io.DataOutputStream confirmation`
Private.
`java.net.Socket connectionSocket`
Private.
`java.lang.String correctMessage`
Private.
`java.io.BufferedReader inFromClient`
Private.
`java.io.DataOutputStream outToServer`
Private.
`int port`
Private.
`int port1`
Private.
`java.lang.String request`
Private.
Fields
`java.lang.Thread runner`
Private.
`int time`
Private.
`java.net.ServerSocket welcomesocket`
Private.
Constructor Summary
Constructor and Description
`ClientEIS()`
Default Constructor.

`ClientEIS(java.lang.String threadname)`
Constructor to start the ServerEISthread thread
Method Summary
Modifier and Type Method and Description
`void run()`
Run method.
`boolean throttleCheck(java.lang.String request, double check)`

E.4.4 Application EISI – package EISI – class CloseWindow.Java

Package EISI
Class CloseWindow
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
EISI.CloseWindow
All Implemented Interfaces:
java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class CloseWindow
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

Singleton class. Indicates that the RMIServer is running. Also indicates if an RMI connection has been established.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`javax.swing.JLabel EISIOff`
Public.
`javax.swing.JLabel EISIOn`
Public.
`static CloseWindow instance`
Private.
`javax.swing.JButton quit`
Private.
Constructor Summary
Constructor and Description
`CloseWindow()`
Instantiates the class object and creates the GUI.
Fields
Constructors
Method Summary
Modifier and Type Method and Description
Assigns action to the JButton quit.
`static CloseWindow getInstance()`
Static method for instantiation.

Application EISI – package EISI – class EISICentral.Java

Package EISI

Class EISICentral

java.lang.Object
java.rmi.server.RemoteObject
java.rmi.server.RemoteServer
java.rmi.server.UnicastRemoteObject
EISI.EISICentral

All Implemented Interfaces:
eisistartinterface.EISIStartInterface, java.io.Serializable, java.rmi.Remote

```
public class EISICentral
extends java.rmi.server.UnicastRemoteObject
implements eisistartinterface.EISIStartInterface
```

Includes 2 methods that are bound upon RMI call. An object of the class is instantiated upon activation of the RMI server
but the methods are not executed until reception of RMI request. Also gets an instance of class CloseWindow to show the
GUI informing about the status of the RMI server. One method performs the initialisation sequence after RMI connection.
The second method initiates the unbind process after appropriate RMI call.

Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
See Also:
Serialized Form

Field Summary

Modifier and Type Field and Description
`java.lang.Process p1`
Public.
`int port`
Private.
`java.net.ServerSocket testS`
Private.

Constructor and Description
`EISICentral()`
Assigns port to ServerSocket testS and gets an instance of class CloseWindow.

Method Summary

Modifier and Type Method and Description
`void endEISI()`
Activated by RMI call.
`void startEISI(java.lang.String hostAddress)`
Central method that commences the sequence of start up.

E.4.5 Application EISI – package EISI – class MessageConfiguration.Java

Package EISI
Interface MessageConfiguration
All Known Implementing Classes:
MessageImplementation, ServerEIS

```
public interface MessageConfiguration
```

Interface which provides abstract methods that configure the messages to be sent to the MEOCS based on the response
received by the ServerEIS from EIS.

Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Method Summary
Modifier and Type Method and Description
`java.lang.String concatenateMessage(java.lang.String response)`
Abstract method.
`java.lang.String controlStatusMessage(java.lang.String response)`
Abstract method.
`java.lang.String engineIndicationsMessages(java.lang.String response)`
Abstract method.
`java.lang.String engineStatusMessages(java.lang.String response)`
Abstract method.
`java.lang.String engineWarningMessages(java.lang.String response)`
Abstract method.
`java.lang.String initiateCheckInput()`
Abstract method.
`java.lang.String packCriticalReadings(java.lang.String response)`
Abstract method.
`java.lang.String systemMessages(java.lang.String response)`
Abstract method.

E.4.6 Application EISI – package EISI – class MessageImplementation.Java

Package EISI
Class MessageImplementation
java.lang.Object
EISI.MessageImplementation
All Implemented Interfaces:
MessageConfiguration
Direct Known Subclasses:
ServerEIS
`public class MessageImplementation`
`extends java.lang.Object`
`implements MessageConfiguration`
Implements the MessageConfiguration interface. It configures the bodies of the abstract methods for derivation of the
system and engine messages, and indications. Should the engine or EIS be replaced, only this class must be updated
provided that EIS transmits via TCP
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Constructor Summary
Constructor and Description
`MessageImplementation()`
Method Summary
Modifier and Type Method and Description
`java.lang.String concatenateMessage(java.lang.String response)`
Configures the final message to be sent to the MEOCS.
`java.lang.String controlStatusMessage(java.lang.String response)`
Creates the messages regarding the system condition.
`java.lang.String engineIndicationsMessages(java.lang.String response)`
Creates the messages regarding the engine indications.
`java.lang.String engineStatusMessages(java.lang.String response)`
Creates the messages regarding the engine status.
Constructors
Methods
`java.lang.String engineWarningMessages(java.lang.String response)`
Creates the messages regarding the engine warnings.
`java.lang.String initiateCheckInput()`
Initiates the the value of the string to correct the input.
`java.lang.String packCriticalReadings(java.lang.String response)`
Configures the final message to be sent to the TSS.
`java.lang.String systemMessages(java.lang.String response)`
Creates the messages regarding the system status.

E.4.7 Application EISI – package EISI – class ProcessClosure.Java

Package EISI
Class ProcessClosure
java.lang.Object
EISI.ProcessClosure
`public class ProcessClosure`
`extends java.lang.Object`
Attaches a shutdown hook for the application shutdown sequence to be initiated upon closure.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Constructor Summary
Constructor and Description
`ProcessClosure()`
Method Summary
Modifier and Type Method and Description
`void attachShutDownHook(java.lang.Process p1)`
Attaches a shutdown hook thread.

E.4.8 Application EISI – package EISI – class RefreshServer.Java

Package EISI
Class RefreshServer
java.lang.Object
EISI.RefreshServer
All Implemented Interfaces:
java.lang.Runnable
`public class RefreshServer`
`extends java.lang.Object`
`implements java.lang.Runnable`
Activated upon RMIServer action() method call. Awaits a one off message from the RMI client in order to
unbind the
already bound service.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`int port`
Private.
`java.lang.Thread runner`
Private.
`java.rmi.registry.Registry thisRegistry`
Private.
Constructor Summary
Constructor and Description
`RefreshServer(java.rmi.registry.Registry registry)`
Default constructor.
`RefreshServer(java.lang.String threadname)`
Constructor to start the thread.
Fields
Constructors
Method Summary
Modifier and Type Method and Description
`void run()`
Run method.

E.4.9 Application EISI – package EISI – class RestartInfo.Java

Package EISI
Class RestartInfo
java.lang.Object
EISI.RestartInfo
All Implemented Interfaces:
java.lang.Runnable
`public class RestartInfo`
`extends java.lang.Object`
`implements java.lang.Runnable`
Updates the java WatchDog application about the activation of the RMI server. When the RMI server is not
running,
WatchDog is informed and consequently restarts the server.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis

Field Summary
Modifier and Type Field and Description
`java.io.BufferedReader in`
Private.
`java.io.DataOutputStream out`
Private.
`int port`
Private.
`java.lang.Thread runner`
Private.
Fields
Constructor Summary
Constructor and Description
`RestartInfo()`
Default constructor.
`RestartInfo(java.lang.String threadname)`
Constructor to start the thread.
Method Summary
Modifier and Type Method and Description
`void run()`
Run method.

E.4.10 Application EISI – package EISI – class RMIServer.Java

Package EISI

Class RMIServer

java.lang.Object
EISI.RMIServer

```
public class RMIServer
extends java.lang.Object
```

RMI server. Creates an RMI Registry object and binds it in port 1099 with a dedicated RMI service.

Since:
10-4-2014

Version:
1.1

Author:
Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`java.rmi.registry.Registry registry`
Private.

`java.lang.String serviceName`
Public.

Constructor Summary

Constructor and Description

`RMIServer()`

Method Summary

Modifier and Type Method and Description

`void action()`
Private method to bind the RMI service.

`static void main(java.lang.String[] args)`
Main method.

E.4.11 Application EISI – package EISI – class ServerEIS.Java

Package EISI

Class ServerEIS

java.lang.Object
EISI.MessageImplementation
EISI.ServerEIS

All Implemented Interfaces:
MessageConfiguration, java.lang.Runnable

```
public class ServerEIS
extends MessageImplementation
implements java.lang.Runnable
```

Receives engine and system status data from the EIS, calls the methods of class MessageConfiguration
and consequently
sends configured message to the mEOCS. Also sends critical parameters values to the TSS.

Since:
10-4-2014

Version:
1.1

Author:
Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`java.net.InetAddress addr`
Private.

`java.lang.String checkEisInput`
Private.

`java.lang.String hostIpAddress`
Private.

`java.io.BufferedReader inFromClient`
Private.

`java.io.BufferedReader inFromMEOCS`
Private.

`java.io.BufferedReader inFromTSS`
Private.

`java.io.DataOutputStream outToClient`
Private.

`java.io.DataOutputStream outToMEOCS`
Private.

`java.io.DataOutputStream outToTSS`
Private.

Fields

`int port`

Private.
`int port1`
Private.
`int port2`
Private.
`java.lang.String response`
Private.
`java.lang.Thread runner`
Private.
Constructor Summary
Constructor and Description
`ServerEIS(java.lang.String hostAddress)`
Default Constructor.
`ServerEIS(java.lang.String threadname, java.lang.String hostAddress)`
Constructor to start the ServerEISthread thread
Method Summary
Modifier and Type Method and Description
`protected void attachShutDownHook(java.net.Socket transmitMessage)`
Executed upon closure.
`void run()`
Implementation of method run() from Runnable interface.
Methods inherited from class EISI.MessageImplementation
`concatenateMessage, controlStatusMessage, engineIndicationsMessages, engineStatusMessages, engineWarningMessages, initiateCheckInput, packCriticalReadings, systemMessages`

E.4.12 Application EISI – package EISI – class TSSCall.Java

Package EISI
Class TSSCall
java.lang.Object
EISI.TSSCall
All Implemented Interfaces:
java.lang.Runnable
```
public class TSSCall
extends java.lang.Object
implements java.lang.Runnable
```
Instantiates a thread which calls TSS.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`java.lang.Thread runner`
Private.
Constructor Summary
Constructor and Description
`TSSCall()`
Default constructor
`TSSCall(java.lang.String threadname)`
Constructor to start the ServerEIStread.
Method Summary
Modifier and Type Method and Description
Fields
Constructors
Methods
`void run()`
Run method.

E.4.13 Application MEOCS – package eisistart (LAN version) – class CamReception.Java

Package eisistart
Class CamReception
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
eisistart.CamReception
All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants
```
public class CamReception
extends javax.swing.JFrame
implements java.lang.Runnable
```
Instantiates a Thread that receives and displays live video image. This class is called only in LAN use. It imports the javacv library.

Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`int bytes`
Private.
`CanvasFrame canvas`
Private.
`java.awt.image.BufferedImage img`
Private.
`int port`
Private.
`java.lang.Thread runner`
Private.
Constructor Summary
Constructor and Description
Fields
Constructors
`CamReception()`
Configures the position and properties of the video canvas frame.
`CamReception(java.lang.String threadname)`
Constructor to start the camReceptionThread.
Method Summary
Modifier and Type Method and Description
`void run()`
Run method.

E.4.14 Application MEOCS – package eisistart (LAN version) – class CommandSender.Java

Package eisistart
Class CommandSender
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
eisistart.CommandSender
All Implemented Interfaces:
java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.event.ChangeListener, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class CommandSender
extends javax.swing.JFrame
implements java.awt.event.ActionListener, javax.swing.event.ChangeListener,
java.lang.Runnable
```

Runs on a dedicated thread. Receives a Userinterface object upon instantiation and adds functionality to sliders and buttons. These are then used to transform the inputs of the controls to commands for the gas turbine, which are transmitted to the EISI ClirntEIS thread.
Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Field Summary
`int command`
Private.
`java.text.NumberFormat nf`
Private.
`java.io.DataOutputStream outToServer`
Private.
`java.lang.Thread runner`
Private.
`java.net.Socket sender`
Private.
`javax.swing.JSlider source`
Private.
`java.lang.Object source2`
Private.
`java.lang.String stringCommand`
Private.
`javax.swing.JToggleButton thisBatterySwitch`
Private.
`javax.swing.JSlider thisStartSwitch`
Private.
`javax.swing.JSlider thisThrottle`
Private.
`int throttleCheck`
Private.

Constructor Summary
CommandSender(UserInterface operationPanel)
Instantiates a CommandSender object.
Method Summary
Modifier and Type Method and Description
void actionPerformed(java.awt.event.ActionEvent ev)
Defines action upon action of the toggle button thisBatterySwitch.
void run()
Run method.
void stateChanged(javax.swing.event.ChangeEvent e)
Defines action upon state change of the sliders.

E.4.15 Application MEOCS – package eisistart (LAN version) – class InUseMessage.Java

Package eisistart
Class InUseMessage
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
eisistart.InUseMessage
All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants
public class **InUseMessage**
extends javax.swing.JFrame
Creates a notice dialogue box when the application is already running. Used only in LAN version.
Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Constructor Summary
Constructor and Description
InUseMessage()
Creates a dialogue box upon instantiation of this class.
Constructor Detail
InUseMessage
public InUseMessage()
Creates a dialogue box upon instantiation of this class. The message displayed is:
"REQUESTED ENGINE CURRENTLY IN USE OR CONNECTION NOT POSSIBLE - PLEASE TRY LATER"

E.4.16 Application MEOCS – package eisistart (LAN version) – class MEOCS.Java

Package eisistart
Class MEOCS
java.lang.Object
eisistart.MEOCS
public class **MEOCS**
extends java.lang.Object
Initiates the sequence of MEOCS start up. Also performs a handshake connection with the EISI upon start, to verify that
the application is available to connect.
Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static int port
Private.
Constructor Summary
Constructor and Description
MEOCS()
Method Summary
Modifier and Type Method and Description
static void main(java.lang.String[] args)
Main method, initiates the start up sequence of MEOCS.

E.4.17 Application MEOCS – package eisistart (LAN version) – class MessageReceiver.Java

Package eisistart
Class MessageReceiver
java.lang.Object
eisistart.MessageReceiver
All Implemented Interfaces:
java.lang.Runnable
`public class MessageReceiver`
`extends java.lang.Object`
`implements java.lang.Runnable`
Runs on a dedicated thread. Receives a Userinterface object upon instantiation and sets the indications
in the text fields
and text areas, according to the continuous input from EISI serverEIS Thread.
Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`java.net.Socket connect`
Public.
`protected java.lang.String engineOutput`
Protected.
`java.io.BufferedReader inComing`
Private.
`protected java.util.Map<java.lang.String,java.lang.Double> indications`
Protected.
`java.lang.String inputCorrection`
Private.
`protected java.util.List<java.lang.String> message`
Protected.
`UserInterface op`
Private.
`java.io.DataOutputStream outGoing`
Private.
`int port`
Private.
`java.net.ServerSocket reception`
Public.
Fields
`java.lang.Thread runner`
Private.
`java.util.Scanner sc`
Private.
`java.util.Scanner scan`
Private.
Constructor Summary
Constructor and Description
`MessageReceiver(java.lang.String threadname)`
Constructor to start the messageReceptionThread.
`MessageReceiver(UserInterface operationPanel)`
Instantiates a MessageReceiver object.
Method Summary
Modifier and Type Method and Description
`void run()`
Run method.
`java.util.Map storeIndications(java.util.List<java.lang.String> message`
Stores the engine parameter values included in the 5th group of information from the
received message.
`java.util.List storeMessage(java.lang.String engineOutput)`
Stores the groups of information included in the received message.

E.4.18 Application MEOCS – package eisistart (LAN version) – class RMIClient.Java

Package eisistart

Class RMIClient

java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
eisistart.RMIClient

All Implemented Interfaces:

java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

```
public class RMIClient
extends javax.swing.JFrame
implements java.awt.event.ActionListener
```

Contains methods that call remote methods via RMI. One method for starting the remote application and one for ending it.

Since:

10-4-2014

Version:

1.1 LAN

Author:

Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`javax.swing.JButton exit`
Private.

`java.lang.String hostAddress`
Private.

`eisistartinterface.EISIStartInterface impl`
Private.

`java.rmi.registry.Registry myRegistry`
Private.

Constructor Summary

Constructor and Description

`RMIClient(UserInterface operationPanel, java.lang.String ipAddress)`
Assigns variables to attributes hostAddress and operationPanel.

Fields

Constructors

Method Summary

Modifier and Type Method and Description

`void action()`
Calls the remote method to start EISI.

`void actionPerformed(java.awt.event.ActionEvent e)`
Defines action to the JButton exit.

`protected void attachShutDownHook()`
Activated upon closure.

`protected void attachShutDownHook2()`
Activated upon closure.

E.4.19 Application MEOCS – package eisistart (LAN version) – class TSSInfo.Java

Package eisistart

Class TssInfo

java.lang.Object

eisistart.TssInfo

All Implemented Interfaces:

java.lang.Runnable

```
public class TssInfo
extends java.lang.Object
implements java.lang.Runnable
```

Runs on a dedicated thread. Receives a Userinterface object upon instantiation and sets the indications in the text fields
and text areas, according to the continuous input from EISI AmbientIn Thread.

Since:

10-4-2014

Version:

1.1 LAN

Author:

Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`java.net.Socket connect`

Private.

`java.io.BufferedReader inComing`

Private.

`UserInterface op`

Private.

`java.io.DataOutputStream outGoing`

Private.

`int port`

Private.

`java.net.ServerSocket reception`

Private.

`java.lang.Thread runner`

Private.

`java.lang.String tssCorrection`

Private.

`protected java.lang.String tssInput`

Protected.

Fields

Constructor Summary

Constructor and Description

`TssInfo(java.lang.String threadname)`

Constructor to start the TssInfoThread.

`TssInfo(UserInterface operationPanel)`

Instantiates a TssInfo object.

Method Summary

Modifier and Type Method and Description

`void run()`

Run method.

E.4.20 Application MEOCS – package eisistart (LAN version) – class UserInterface.Java

Package eisistart
Class UserInterface
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
eisistart.UserInterface
All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants

`public class UserInterface`
`extends javax.swing.JFrame`

Singleton class.Instantiates a GUI that represents the operation panel of the remote gas turbine. Used only in LAN version.
This class is tightly coupled to the particular gas turbine. In a future generic application, there will be a prototype class cloned
according to the gas turbine selected from the user (Prototype Pattern).
Since:
10-4-2014
Version:
1.1 LAN
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`javax.swing.JTextField ambientPresField`
Public.
`javax.swing.JTextField ambientTempField`
Public.
`javax.swing.JToggleButton batterySwitch`
Public.
`javax.swing.JTextArea controlStatusField`
Public.
`javax.swing.JLabel egtAbs`
Public.
`javax.swing.JTextField egtField`
Public.
`javax.swing.JSlider egtIndicator`
Public.
`javax.swing.JLabel egtRel`
Public.
`javax.swing.JTextArea errorsIndicationField`
Public.
`javax.swing.JTextField fflowField`
Public.
`protected java.lang.String inputChecker`
Protected.
`static UserInterface instance`
Private.
`javax.swing.JLabel powerSupplyIndicator`
Public.
`javax.swing.JLabel rpmAbs`
Public.
`javax.swing.JTextField rpmField`
Public.
`javax.swing.JSlider rpmIndicator`
Public.
`javax.swing.JLabel rpmRel`
Public.
`javax.swing.JSlider startSwitch`
Public.
`javax.swing.JTextArea statusIndicationField`
Public.
`javax.swing.JButton stopProgram`
Public.
`javax.swing.JTextField supplyField`
Public.
`javax.swing.JTextArea systemStatusIndicationField`
Public.

Fields
`javax.swing.JSlider throttle`
Public.
`javax.swing.JTextField throttleFeedback`
Public.
`protected java.lang.String tssInputChecker`
Protected.
`javax.swing.JTextField voutField`
Public.
Constructor Summary
Constructor and Description
`UserInterface()`
Creates the GUI that represents the AMT Netherlands Olympus gas turbine operation panel.
Method Summary
Modifier and Type Method and Description
`static UserInterface getInstance()`
Static method for instantiation.
`void setAreas(java.util.List<java.lang.String> message)`
Updates the text areas.
`void setIndications(java.util.Map<java.lang.String,java.lang.Double> indications)`
Updates the text fields and the indication sliders.
`void setTssInfo(java.lang.String tssInput)`
Updates the text fields related with ambient info and the Caution Panel JLabels.
ring message package received from EISI AmbientIn Thread.

E.4.21 Application MEOCS – package start (WEB version) – class CommandSender.Java

Package start
Class CommandSender
java.lang.Object
start.CommandSender
All Implemented Interfaces:
java.lang.Runnable
public class **CommandSender**
extends java.lang.Object
implements java.lang.Runnable
Includes static methods that are called by web services. Its main purpose is to configure the command received from the
web service and send it to the EISI in the expected form.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static java.io.BufferedReader confirmation
Private.
static java.io.DataOutputStream outToServer
Private.
int port
Private.
static java.lang.Thread runner
Private.
static java.net.Socket sender
Private.
Fields
Constructor Summary
Constructor and Description
CommandSender()
Constructor to start the enginePanelThread.
Method Summary
Modifier and Type Method and Description
static void closePort()
Closes the TCP Socket.
void run()
Instantiates the TCP Socket, the DataOutputStream and the BufferedReader.
static java.lang.String stateChanged(double flag, double command)
Configures and sends the command to the EISI.

E.4.22 Application MEOCS – package start (WEB version) – class MEOCS.Java

Package start
Class MEOCS
java.lang.Object
start.MEOCS
public class **MEOCS**
extends java.lang.Object
Initiates the sequence of MEOCS start up. Also performs a handshake connection with the EISI upon start, to verify that
the application is available to connect.

Since:
10-4-2014

Version:
1.1 web

Author:
Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

static int port
Private.
static int port2
Private.
static RMIClient rmic
Private.

Constructor Summary

Constructor and Description

MEOCS()

Method Summary

Modifier and Type Method and Description

Fields

Constructors

Methods

static java.lang.String main()
Initiates the start up sequence of MEOCS.
static void release()
Creates a TCP Socket that sends a one off message to the remote RMI server to unbind the service and Registry.

E.4.23 Application MEOCS – package start (WEB version) – class MessageReceiver.Java

Package start
Class MessageReceiver
java.lang.Object
start.MessageReceiver
All Implemented Interfaces:
java.lang.Runnable
public class **MessageReceiver**
extends java.lang.Object
implements java.lang.Runnable
Includes a static method that is called by a web service. Its main purpose is to receive engine and system status data from
the EISI and return them to the root resource that requested them, in the expected form. During instantiation it declares
and instantiates the required sockets.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static java.net.Socket **connect**
Public.
protected static java.lang.String **engineOutput**
Protected.
static java.io.BufferedReader **inComing**
Private.
static java.lang.String **inputCorrection**
Private.
static java.io.DataOutputStream **outGoing**
Private.
static int **port**
Private.
static java.net.ServerSocket **reception**
Public.
java.lang.Thread **runner**
Private.
Fields
Constructor Summary
Constructor and Description
MessageReceiver()
Default constructor
MessageReceiver(java.lang.String threadname)
Constructor to start the messageReceptionThread.
Method Summary
Modifier and Type Method and Description
static java.lang.String readEngData()
Reads input from serverEIS Thread of EISI.
void run()
Run method.

E.4.24 Application MEOCS – package start (WEB version) – class RMIClient.Java

Package start
Class RMIClient
java.lang.Object
start.RMIClient
public class **RMIClient**
extends java.lang.Object
Contains methods that call remote methods via RMI. One method for starting the remote application and one for ending it.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.lang.String **hostAddress**
Private.
eisistartinterface.EISIStartInterface **impl**
Private.
java.rmi.registry.Registry **myRegistry**
Private.
Constructor Summary
Constructor and Description
RMIClient(java.lang.String ipAddress)
Default constructor.
Method Summary
Modifier and Type Method and Description
Fields
Constructors
Methods
void action()
Calls the remote method to start EISI.
void stopRemote()
Calls the remote method to shutdown EISI.

E.4.25 Application MEOCS – package start (WEB version) – class TSSInfo.Java

Package start
Class TssInfo
java.lang.Object
start.TssInfo
All Implemented Interfaces:
java.lang.Runnable
public class **TssInfo**
extends java.lang.Object
implements java.lang.Runnable
Includes a static method that is called by a web service. Its main purpose is to receive ambient, peripheral and
exceedance data from the EISI and return them to the root resource that requested them, in the expected form. During
instantiation it declares and instantiates the required sockets.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static java.net.Socket **connect**
Public.
static java.io.BufferedReader **inComing**
Private.
static java.io.DataOutputStream **outGoing**
Private.
int **port**
Private.
static java.net.ServerSocket **reception**
Public.
java.lang.Thread **runner**
Private.
static java.lang.String **tssCorrection**
Private.
protected static java.lang.String **tssInput**
Protected.
Fields
Constructor Summary
Constructor and Description

E.4.26 Application MEOCS and EISI – package eisistartinterface (Common RMI interface) – interface EISIStartInterface.Java

Package eisistartinterface
Interface EISIStartInterface
All Superinterfaces:
java.rmi.Remote
`public interface EISIStartInterface`
`extends java.rmi.Remote`
Common Interface with RMI client.
Since:
10-4-2014
Version:
1.1 web
Author:
Michael Diakostefanis
Method Summary
Modifier and Type Method and Description
`void endEISI()`
Abstract method.
`void startEISI(java.lang.String hostAddress)`
Abstract method.
Method Detail
startEISI
`void startEISI(java.lang.String hostAddress)`
`throws java.rmi.RemoteException`
Abstract method. Calls the method that initiates the EISI startup sequence.
Parameters:
`hostAddress -`
Throws:
`java.rmi.RemoteException`
endEISI
Methods
`void endEISI()`
`throws java.rmi.RemoteException`
Abstract method. Calls the method that initiates the EISI shutdown sequence.
Throws:
`java.rmi.RemoteException`

E.4.27 Application TSS – package tss – class AmbientInput.Java

Package tss
Class AmbientInput
java.lang.Object
tss.AmbientInput
All Implemented Interfaces:
java.lang.Runnable
public class **AmbientInput**
extends java.lang.Object
implements java.lang.Runnable
Receives ambient and peripheral data from LabVIEW ACAS module.Data is received by TCP and stored in
a HashMap and then passed to class
Compare input.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.lang.String **ACASInput**
Private.
protected java.util.Map<java.lang.String,java.lang.Double> **AmbientSet**
Protected.
java.lang.String **check**
Private.
java.util.concurrent.Exchanger **ex**
Private.
java.util.Scanner **filter**
Private.
int **port**
Private.
java.lang.String **r**

Private.
java.lang.Thread **runner**
Private.
java.util.Scanner **scan**
Private.
java.util.Map<java.lang.String,java.lang.Double> **toBeSent**
Private.
Constructor Summary
Constructor and Description
AmbientInput(java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Double>> c
Assigns Exchanger ex to the instance of Exchanger passed from method TSSCentral.tssMain.
Fields
Constructors
AmbientInput(java.lang.String threadname)
Constructor to start the AmbientInputThread thread.
Method Summary
Modifier and Type Method and Description
protected java.lang.String **filterInput**(java.lang.String ACASInput)
Filters the received input from ACAS.
void **run**()
Run method.
protected java.util.Map **setAmbientValues**(java.lang.String r)
Stores the ambient and peripheral parameters values.

E.4.28 Application TSS – package tss – class AutomaticEngineShutdown.Java

Package tss
Class AutomaticEngineShutdown
java.lang.Object
tss.AutomaticEngineShutdown
All Implemented Interfaces:
java.lang.Runnable
public class **AutomaticEngineShutdown**
extends java.lang.Object
implements java.lang.Runnable
This class performs 2 major tasks:
1. Sends an emergency signal to EIS to shutdown the engine
2. Terminates the execution of TSS and EISI by exiting the current instance of the JVM.
The above tasks are always performed sequentially when this class detects the following:
1. Absolute value exceedance of RPM or EGT has been detected
2. Mains supply power loss for more than 3 minutes
3. Relative value exceedance of RPM or EGT holds for more than 3 minutes.
Runs on a dedicated thread.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.util.Map<java.lang.String,java.lang.Boolean> **booleanMap**
Private.
long **elapsedTime**
Private.
long **elapsedTime2**
Private.
java.util.concurrent.Exchanger **exchn**
Private.
long **loopTime**
Private.
long **loopTime2**
Private.
int **port**
Private.
java.lang.Thread **runner**
Private.
long **startTime**
Private.
Constructor Summary
Constructor and Description
AutomaticEngineShutdown(java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Bool
ean>> b)
Default constructor.
Fields
Constructors
AutomaticEngineShutdown(java.lang.String threadname)
Constructor to start the AutomaticEngineShutdownThread.
Method Summary
Modifier and Type Method and Description
void **emergencyMessage**()
This method is called only in case of emergency.
void **run**()
Run method.

E.4.29 Application TSS – package tss – class CompareInput.Java

Package tss
Class CompareInput
java.lang.Object
tss.CompareInput
public class **CompareInput**
extends java.lang.Object
Receives the nominal values, engine values and ambient values from the other classes of TSS and searches for absolute or relative
exceedance, after correction of the engine parameters to ISASL conditions. It informs class AutomaticEngineShutdown which takes
appropriate action.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.util.Map<java.lang.String,java.lang.Boolean> **booleanMap**
Private.
boolean **compareMode**
Private.
int **EGTabsolute**
Private.
int **EGTLimit**
Private.
java.net.Socket **eisiClient**
Private.
long **elapsedTime**
Private.
long **endTime**
Private.
java.util.Map<java.lang.String,java.lang.Double> **engIn**
Protected.
java.util.concurrent.Exchanger **ex**
Private.
java.util.concurrent.Exchanger **ex2**
Private.
java.util.concurrent.Exchanger **ex3**
Private.
java.io.BufferedReader **inFromEISI**
Private.
int **iteration**
Private.
int **mainsOut**
Private.
Fields
java.io.DataOutputStream **outToEISI**
Private.
int **port**
Private.
ReadNominals rn
Public.
java.util.List<java.util.Map> **rnom**
Private.
int **RPMabsolute**
Private.
int **RPMLimit**
Private.
long **startTime**
Private.
java.util.Map<java.lang.String,java.lang.Double> **str**
Private.
java.util.List<java.lang.Integer> **throttleCheck**
Private.
int **throttleReference**
Private.
Constructor Summary
Constructor and Description
CompareInput(ReadNominals rn,
java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Double>> c,
java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Double>> d,
java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Boolean>> f)
Assigns Exchanger ex, ex2 and ex3 to the instance of Exchanger passed from method TSSCentral.tssMain.
Method Summar
Modifier and Type Method and Description
int **absoluteEGT**(java.util.Map<java.lang.String,java.lang.Double> engIn)
Checks the absolute limit of EGT.
int **absoluteRPM**(java.util.Map<java.lang.String,java.lang.Double> engIn)
Checks the absolute limit of RPM.

void compare()
Co-ordinates the check of the engine input values and mains power supply.
java.lang.String infoToEISI(int mainsOut, int EGTabsolute, int RPMabsolute, int EGTLimit,
int RPMLimit, java.util.Map<java.lang.String,java.lang.Double> str)
Configure the message sent back to EISI.
int mainsSupply(java.util.Map<java.lang.String,java.lang.Double> str)
Checks mains power supply.
int relativeEGT(java.util.List<java.util.Map> rnom,
java.util.Map<java.lang.String,java.lang.Double> engIn,
java.util.Map<java.lang.String,java.lang.Double> str)
Checks relative exceedance of EGT.
int relativeRPM(java.util.List<java.util.Map> rnom,
java.util.Map<java.lang.String,java.lang.Double> engIn,
java.util.Map<java.lang.String,java.lang.Double> str)
Checks relative exceedance of RPM.
java.util.Map setBooleans(int mainsOut, int EGTabsolute, int RPMabsolute, int EGTLimit,
int RPMLimit, boolean compareMode)
Configures the boolean variables to be sent to the AutomaticEngineShutdown class.

E.4.30 Application TSS – package tss – class EngineInput.Java

Package tss
Class EngineInput
java.lang.Object
tss.EngineInput
All Implemented Interfaces:
java.lang.Runnable
public class **EngineInput**
extends java.lang.Object
implements java.lang.Runnable
Reads a String message from EISI with the engine readings. The parameter values are then stored in a
Hash Map and provided to class
CompareInput.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.lang.String check
Private.
java.lang.String EISIInput
Private.
protected java.util.Map<java.lang.String,java.lang.Double> engineSet
Protected.
java.util.concurrent.Exchanger ex
Private.
java.util.Scanner filter
Private.
int port
Private.
java.lang.String r
Private.
java.lang.Thread runner
Private.
java.util.Scanner scan
Private.
java.util.Map<java.lang.String,java.lang.Double> toBeSent
Private.
Constructor Summary
Constructor and Description
EngineInput(java.util.concurrent.Exchanger<java.util.Map<java.lang.String,java.lang.Double>> c)
Assigns Exchanger ex to the instance of Exchanger passed from method TSSCentral.tssMain.
Fields
Constructors
EngineInput(java.lang.String threadname)
Constructor to start the EngineInputThread.
Method Summary
Modifier and Type Method and Description
protected java.lang.String filterEISIInput(java.lang.String engineInput)
Filters the received input from EISI.
void run()
Run method.
protected java.util.Map setEngineValues(java.lang.String r)
Stores the critical engine parameters values.

E.4.31 Application TSS – package tss – class ReadNominals.Java

Package tss
Class ReadNominals
java.lang.Object
tss.ReadNominals
public class **ReadNominals**
extends java.lang.Object
Rads the nominals.txt file with the nominal values of critical parameters RPM and EGT along with their
standard deviation
for each throttle setting. The values are then stored in an ArrayList.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
java.io.BufferedReader **br**
Private.
java.io.File **file**
Private.
java.lang.String **line**
Private.
protected java.util.List<java.util.Map> **nominals**
Protected.
java.io.FileReader **nominalsFile**
Private.
java.util.Scanner **scan**
Private.
protected java.util.Map<java.lang.String,java.lang.Double> **settingValues**
Protected.
Constructor Summary
Constructor and Description
ReadNominals()
Fields
Constructors
Method Summary
Modifier and Type Method and Description
java.util.List **storeNominals()**
Reads the file with the nominal values and stores them in an ArrayList of HashMaps.

E.4.32 Application TSS – package tss – class TSSCentral.Java

Package tss
Class TSSCentral
java.lang.Object
tss.TSSCentral
public class **TSSCentral**
extends java.lang.Object
Central class of TSS. Instantiates all the threads and required objects.
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Constructor Summary
Constructor and Description
TSSCentral()
Method Summary
Modifier and Type Method and Description
static void **tssMain()**
Initiates the start up sequence of TSS.

E.4.33 Application RestartServer – package restartserver – class RestartServer.Java

Package restartserver
Class RestartServer
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
restartserver.RestartServer
All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible,
javax.swing.RootPaneContainer, javax.swing.WindowConstants

public final class **RestartServer**
extends javax.swing.JFrame
Singleton class. An intermediate application that is triggered from WatcDog program (which consequently closes
programmatically) when the RMIServer is not running. The purpose of this program is to restart WatchDog application to
monitor the status of the RMIServer.
Since:
10-4-2014
Version:
1.1 master
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static RestartServer **instance**
private.
Constructor Summary
Constructor and Description
RestartServer()
Creates a RestartServer object with the GUI.
Method Summary
Modifier and Type Method and Description
Fields
Constructors
Methods
static RestartServer **getInstance()**
Static method for instantiation.
static void **main**(java.lang.String[] args)

Main method.

E.4.34 Application WatchDog – package watchdog – class WatchDog.Java

Package watchdog
Class WatchDog
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
watchdog.WatchDog
All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, javax.accessibility.Accessible,
javax.swing.RootPaneContainer, javax.swing.WindowConstants
public final class **WatchDog**
extends javax.swing.JFrame
Singleton class. This class contains a main method that instantiates a Socket to monitor the operation of the EISI
RMIServer.
Since:
10-4-2014
Version:
1.1 master
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
static java.io.BufferedReader **in**
private.
static WatchDog **instance**
private.
static java.io.DataOutputStream **out**
private.
static int **port**
private.
static java.net.Socket **soc**
private.
Constructor Summary
Constructor and Description
Fields
Constructors
WatchDog()
Creates a WatchDog object with the GUI
Method Summary
Modifier and Type Method and Description
static WatchDog **getInstance()**
Static method for instantiation.
static void **main**(java.lang.String[] args)
Main method.

E.4.35 Application InternetGasturbine – package eisistart – class CommandResource.Java

Package eisistart
Class CommandResource
java.lang.Object
eisistart.CommandResource
`public class CommandResource`
`extends java.lang.Object`
REST Web Service. Accepts requests from javascript client for commands to the remote gas turbine. Path: "command/{a}/{b}"
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`private javax.ws.rs.core.UriInfo context`
Constructor Summary
Constructor and Description
`CommandResource()`
Creates a new instance of CommandResource
Method Summary
Modifier and Type Method and Description
`java.lang.String getText(double a, double b)`
GET method to retrieve representation of an instance of eisistart.CommandResource.
`java.lang.String postText(double a, double b)`
POST method for updating an instance of CommandResource.
`java.lang.String putText(double a, double b)`
PUT method for updating or creating an instance of CommandResource.

E.4.36 Application InternetGasturbine – package eisistart – class LogoutResource.Java

Package eisistart
Class LogoutResource
java.lang.Object
eisistart.LogoutResource
`public class LogoutResource`
`extends java.lang.Object`
REST Web Service. Accepts requests from javascript client for user logout. Path: "logout"
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`private javax.ws.rs.core.UriInfo context`
Constructor Summary
Constructor and Description
`LogoutResource()`
Creates a new instance of LogoutResource
Method Summary
Modifier and Type Method and Description
`com.sun.jersey.api.view.Viewable getHtml()`
GET method to retrieve representation of an instance of eisistart.LogoutResource.
`com.sun.jersey.api.view.Viewable postHtml()`
POST method for updating an instance of LogoutResource.
`com.sun.jersey.api.view.Viewable putHtml()`
PUT method for updating or creating an instance of LogoutResource.

E.4.37 Application InternetGasturbine – package eisistart – class ReadAmbResource.Java

Package eisistart
Class ReadAmbResource
java.lang.Object
eisistart.ReadAmbResource
`public class ReadAmbResource`
`extends java.lang.Object`
REST Web Service. Accepts requests from javascript client for ambient and warning data acquisition from the remote gas
turbine. Path:
"readAmb"

Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis
Field Summary
Modifier and Type Field and Description
`private javax.ws.rs.core.UriInfo context`
Constructor Summary
Constructor and Description
`ReadAmbResource()`
Creates a new instance of ReadAmbResource
Method Summary
Modifier and Type Method and Description
`java.lang.String getText()`
GET method to retrieves representation of an instance of
eisistart.ReadAmbResource.
`java.lang.String postText()`
POST method for updating an instance of ReadAmbResource.
`java.lang.String putText()`
PUT method for updating or creating an instance of ReadAmbResource.

E.4.38 Application InternetGasturbine – package eisistart – class ReadEngResource.Java

Package eisistart
Class ReadEngResource
java.lang.Object
eisistart.ReadEngResource
`public class ReadEngResource`
`extends java.lang.Object`
REST Web Service. Accepts requests from javascript client for engine and
system status data acquisition from the remote
gas turbine. Path:
"readEng"
Since:
10-4-2014
Version:
1.1
Author:
Michael Diakostefanis, Michael-Ria
Field Summary
Modifier and Type Field and Description
`private javax.ws.rs.core.UriInfo context`
Constructor Summary
Constructor and Description
`ReadEngResource()`
Creates a new instance of EnginedataResource
Method Summary
Modifier and Type Method and Description
`java.lang.String getText()`
GET metgod to retrieve representation of an instance of
eisistart.EnginedataResource.
`java.lang.String postText()`
POST method for updating an instance of EnginedataResource.
`java.lang.String putText()`
PUT method for updating or creating an instance of EnginedataResource.

E.4.39 Application InternetGasturbine – package eisistart – class RunResource.Java

Package eisistart

Class RunResource

java.lang.Object
eisistart.RunResource

`public class RunResource`
`extends java.lang.Object`

REST Web Service. Accepts requests from javascript client for the operation panel page of the remote gas turbine. Path:
"run"

Since:
10-4-2014

Version:
1.1

Author:
Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`private javax.ws.rs.core.UriInfo context`

Constructor Summary

Constructor and Description

`RunResource()`
Creates a new instance of RunResource

Method Summary

Modifier and Type Method and Description

`java.lang.String getHtml()`
GET method to retrieve representation of an instance of eisistart.RunResource.

`com.sun.jersey.api.view.Viewable postHtml()`
POST method for updatingan instance of RunResource.

`com.sun.jersey.api.view.Viewable putHtml()`
PUT method for updating or creating an instance of RunResource.

E.4.40 Application InternetGasturbine – package eisistart – class StopResource.Java

Package eisistart

Class StopResource

java.lang.Object
eisistart.StopResource

`public class StopResource`
`extends java.lang.Object`

REST Web Service. Accepts requests from javascript client for termination of the application of remote gas turbine
operation. Path:
"stop"

Since:
10-4-2014

Version:
1.1

Author:
Michael Diakostefanis

Field Summary

Modifier and Type Field and Description

`private javax.ws.rs.core.UriInfo context`

Constructor Summary

Constructor and Description

`StopResource()`
Creates a new instance of StopResource

Method Summary

Modifier and Type Method and Description

`java.lang.String getText()`
GET method to retrieve representation of an instance of eisistart.StopResource.

`java.lang.String putText()`
POST method for updating or creating an instance of StopResource.

`java.lang.String putXml(java.lang.String content)`
PUT method for updating or creating an instance of StopResource.

E.5 JavaScript files - interface.js

File Index
interface.js
This file is associated with interface.jsp. It provides necessary functionality.
Author:
Michael Diakostefanis
Version:
1.1
Since:
10-4-2014

Field Summary
amb
Executed upon loading of the page.
button2
Instance of Y.Button.
button3
Instance of Y.Button.
button4
Instance of Y.Button.
button5
Instance of Y.Button.
button6
Instance of Y.Button.
buttonGroupRadio
Instance of Y.ButtonGroup.
buttonGroupSwitch
Instance of Y.ButtonGroup.
chart
Google visualisation API, chart type: Gauge.
chart1
Google visualisation API, chart type: Gauge.
check
Global variable initialised to value 1.
data
A google visualisatio Data Table.
data1
A google visualisatio Data Table.
eng
Executed upon loading of the page.
throttleSlider
Configures the throttle slider.
xmlhttp1
POST Request sent when button2 is pressed (Battery ON).

xmlhttp10
POST Request sent when function readAmbient()is called.
xmlhttp2
POST Request sent when button3 is pressed (Battery OFF).
xmlhttp3
POST Request sent when button4 is pressed (Switch STOP).
xmlhttp4
POST Request sent when button5 is pressed (Switch AUTO - STOP).
xmlhttp5
POST Request sent when button6 is pressed (Switch RUN).
xmlhttp6
POST Request sent when slider value is changed manually.
xmlhttp7
POST Request sent when slider value is changed by the mouse.
xmlhttp8
GET Request sent when function quitPage() is called.
xmlhttp9
GET Request sent when function logoutPage() is called.
xmlhttped
POST Request sent when function engineData() is called.
yuiButtons
Configures the control buttons.
Method Summary
batteryOff()
Activated on click of button3.
batteryOn()
Activated on click of button2.
drawChart()
Draws a gauge for the EGT tndication of the remote gas turbine.
drawChart1()
Draws a gauge for the RPM tndication of the remote gas turbine.
engineData()
Creates a POST HTTP request to a URL of a REST web service to invoke engine and system status data acquisition from the server side application.
logoutPage()
Function activated upon logout selection from the operation panel page.
quitPage()
Function activated upon closure of the page.
readAmbient()
Creates a POST HTTP request to a URL of a REST web service to invoke ambient and warning data acquisition from the server side application.
setValueEgt(egt)
Called within function engineData, when the page is on staedy state.
setValueRpm(rpm)
Called within function engineData, when the page is on staedy state.
sliderFieldChange()
Function to update the slider when the value of the associated text field has changed.
switchAutoStop()
Activated on click of button5.
switchRun()
Activated on click of button6.
switchStop()
Activated on click of button4.
updateSlider(e)
Function to update the input value of the text field from the Slider value.

Appendix F Installation Guidelines

F.1 Local Installation

Prerequisites:

- Windows 7, 64 bit
- Java Runtime Environment (JRE) 7 for Windows 7 64 bit
- LabVIEW 2011SP1 Runtime Environment for Windows 7 64 bit
- Opencv version 2.4.5 or later for Windows (Only for LAN version). Environmental variable PATH value set to "C:\opencv\build\x64\vc10\bin\;"
- Wowza Media Server version 3.6.2 or later (Only for web version). The following environmental values set:
  - WMSAPP_HOME: "C:\Program Files (x86)\Wowza Media Systems\Wowza Media Server 3.6.2"
  - WMSCONFIG_HOME: "C:\Program Files (x86)\Wowza Media Systems\Wowza Media Server 3.6.2"
  - WMSINSTALL_HOME: "C:\Program Files (x86)\Wowza Media Systems\Wowza Media Server 3.6.2\"

Procedures:

- Olympus Gas Turbine ECU connected to USB COM3 port
- Barometer connected to USB COM4 port
- DAQ unit connected to any available USB port
- Web Cam connected to any available USB port
- Preserve ports indicated by the provider available
- Extraction of provided .zip file that contains folder "Internet Gas Turbine - Local"
- Start RMI server by executing file WatchDog.jar in sub directiory "/bin"
- For Web version only: Start Wowza media server by executing batch file startup.bat in sub directiory "/bin"

Notes:

- TSS is imported into EISI as an external library upon built
- NominalValues.txt file is included in the "/bin" directory
- Documentation of the modules included in the "/docs" sub directory

Folder "Internet Gas Turbine - Local" PATH listing:

```
C:.
|  tree.txt
|
+---bin
|  |  EIS.aliases
```

```
|   |   EIS.exe
|   |   EIS.ini
|   |   EISI.jar
|   |   NominalValues.txt
|   |   README-EISI.TXT
|   |   README-RestartServer.TXT
|   |   README-WatchDog.TXT
|   |   RestartServer.jar
|   |   SignalClosure.aliases
|   |   SignalClosure.exe
|   |   SignalClosure.ini
|   |   WatchDog.jar
|   |
|   \---lib
|           Javacpp.jar
|           JavaCV-windows-x86_64.jar
|           JavaCV.jar
|           jna-3.5.1.jar
|           README-TSS.TXT
|           TSS.jar
|
+---docs
|   +---EISI docs
|   |   |   allclasses-frame.html
|   |   |   allclasses-noframe.html
|   |   |   constant-values.html
|   |   |   deprecated-list.html
|   |   |   help-doc.html
|   |   |   index-all.html
|   |   |   index.html
|   |   |   overview-tree.html
|   |   |   package-list
|   |   |   serialized-form.html
|   |   |   stylesheet.css
|   |   |
|   |   +---EISI
|   |   |       AmbientIn.html
|   |   |       CamCapture.html
|   |   |       ClientEIS.html
|   |   |       CloseWindow.html
|   |   |       EISICentral.html
|   |   |       MessageConfiguration.html
|   |   |       MessageImplementation.html
|   |   |       package-frame.html
|   |   |       package-summary.html
|   |   |       package-tree.html
|   |   |       ProcessClosure.html
|   |   |       RefreshServer.html
|   |   |       RestartInfo.html
|   |   |       RMIServer.html
|   |   |       ServerEIS.html
|   |   |       TSSCall.html
|   |   |
|   |   +---eisistartinterface
|   |   |       EISIStartInterface.html
|   |   |       package-frame.html
|   |   |       package-summary.html
|   |   |       package-tree.html
|   |   |
|   |   \---resources
|   |           background.gif
|   |           tab.gif
|   |           titlebar.gif
|   |           titlebar_end.gif
|   |
|   +---Monitoring docs
|   |   |   allclasses-frame.html
|   |   |   allclasses-noframe.html
```

```
|   |   |   constant-values.html
|   |   |   deprecated-list.html
|   |   |   help-doc.html
|   |   |   index-all.html
|   |   |   index.html
|   |   |   overview-tree.html
|   |   |   package-list
|   |   |   serialized-form.html
|   |   |   stylesheet.css
|   |   |
|   |   +---resources
|   |   |       background.gif
|   |   |       tab.gif
|   |   |       titlebar.gif
|   |   |       titlebar_end.gif
|   |   |
|   |   \---watchdog
|   |           package-frame.html
|   |           package-summary.html
|   |           package-tree.html
|   |           WatchDog.html
|   |
|   +---Restart Server docs
|   |   |   allclasses-frame.html
|   |   |   allclasses-noframe.html
|   |   |   constant-values.html
|   |   |   deprecated-list.html
|   |   |   help-doc.html
|   |   |   index-all.html
|   |   |   index.html
|   |   |   overview-tree.html
|   |   |   package-list
|   |   |   serialized-form.html
|   |   |   stylesheet.css
|   |   |
|   |   +---resources
|   |   |       background.gif
|   |   |       tab.gif
|   |   |       titlebar.gif
|   |   |       titlebar_end.gif
|   |   |
|   |   \---restartserver
|   |           package-frame.html
|   |           package-summary.html
|   |           package-tree.html
|   |           RestartServer.html
|   |
|   \---TSS docs
|       |   allclasses-frame.html
|       |   allclasses-noframe.html
|       |   constant-values.html
|       |   deprecated-list.html
|       |   help-doc.html
|       |   index-all.html
|       |   index.html
|       |   overview-tree.html
|       |   package-list
|       |   stylesheet.css
|       |
|       +---resources
|       |       background.gif
|       |       tab.gif
|       |       titlebar.gif
|       |       titlebar_end.gif
|       |
|       \---tss
|               AmbientInput.html
|               AutomaticEngineShutdown.html
```

```
|           CompareInput.html
|           EngineInput.html
|           package-frame.html
|           package-summary.html
|           package-tree.html
|           ReadNominals.html
|           TSSCentral.html
|
\---LabVIEW
   +---ACAS
   |      Ambient TCP Client.pdf
   |      MSIS.pdf
   |      Thermocouple.pdf
   |
   +---EIS
   |      EDT Warning signal - Emerg signal.pdf
   |      Fuel flow-Utilities signal.pdf
   |      SerialInput.pdf
   |      SwitchSignal.pdf
   |      TCPClient.pdf
   |      TCPServer.pdf
   |      ThrottleSignal.pdf
   |      Utilities Acquisition.pdf
   |
   \---Signal Reset
          SIGNAL CLOSURE.pdf
```

F.2 LAN client Installation

Prerequisites:

- Java Runtime Environment (JRE) 6 or later

Procedures:

- Extraction of provided .zip file that contains folder "Internet Gas Turbine - Interm - LAN version"
- Preserve ports indicated by the provider available
- Run client by execution of file MEOCS.jar in sub directory "/bin"

Notes:

- JavaCV files are included as an external library
- Documentation of the modules included in the "/docs" sub directory

Folder "Internet Gas Turbine - Interm - LAN version" PATH listing:

```
C:.
|  tree.txt
|
+---bin
|  |  MEOCS.jar
|  |  README.TXT
|  |
|  \---lib
|        Javacpp.jar
|        JavaCV-windows-x86_64.jar
|        JavaCV.jar
|
\---docs
   |  allclasses-frame.html
```

```
|  allclasses-noframe.html
|  constant-values.html
|  deprecated-list.html
|  help-doc.html
|  index-all.html
|  index.html
|  overview-tree.html
|  package-list
|  serialized-form.html
|  stylesheet.css
|
+---eisistart
|      CamReception.html
|      CommandSender.html
|      InUseMessage.html
|      MEOCS.html
|      MessageReceiver.html
|      RMIClient.html
|      TssInfo.html
|      UserInterface.html
|
+---eisistartinterface
|      EISIStartInterface.html
|      package-frame.html
|      package-summary.html
|      package-tree.html
|
\---resources
      background.gif
      tab.gif
      titlebar.gif
      titlebar_end.gif
```

F.3 Web version – Back End Server

Prerequisites:

- Java Runtime Environment (JRE) 7
- Oracle glassfish version 3.1.2.2 or later
- Environmental variable JAVA_HOME set to point to Java directory

Procedures:

- Extraction of provided .zip file that contains folder "Internet Gas Turbine - Interm - web version"
- Start glassfish domain as administrator:

  "asadmin start-domain domain_name"

- Enter administration console of glassfish at port 4848
- Configure glassfish realm with group names and role names as dictated by web.xml and glassfish-web.xml files in sub directory "/WEB-INFO"
- Add new users in configured realm and set usernames and passwords
- Deploy the 2 .war files included in sub directory "/bin":

  "asadmin deploy path_to_file/Internet_Gas_Turbine_Pages.war" “asadmin deploy path_to_file/InternetGasturbine.war”

Notes:

- Application MEOCS is imported in the application InternetGasturbine and included in the .war file during the build
- Documentation of the modules included in the “/docs” sub directory

Folder Internet Gas Turbine - Interm - web version” PATH listing:

```
C:.
|   tree.txt
|
+---bin
|       InternetGasturbine.war
|       Internet_Gas_Turbine_Pages.war
|
+---docs
|   +---InternetGasTurbine
|   |   |   allclasses-frame.html
|   |   |   allclasses-noframe.html
|   |   |   constant-values.html
|   |   |   deprecated-list.html
|   |   |   help-doc.html
|   |   |   index.html
|   |   |   overview-tree.html
|   |   |   package-list
|   |   |   stylesheet.css
|   |   |
|   |   +---eisistart
|   |   |       CommandResource.html
|   |   |       LogoutResource.html
|   |   |       package-frame.html
|   |   |       package-summary.html
|   |   |       package-tree.html
|   |   |       package-use.html
|   |   |       ReadAmbResource.html
|   |   |       ReadEngResource.html
|   |   |       RunResource.html
|   |   |       StopResource.html
|   |   |
|   |   +---index-files
|   |   |       index-1.html
|   |   |       index-2.html
|   |   |       index-3.html
|   |   |       index-4.html
|   |   |       index-5.html
|   |   |       index-6.html
|   |   |       index-7.html
|   |   |
|   |   \---resources
|   |           background.gif
|   |           tab.gif
|   |           titlebar.gif
|   |           titlebar_end.gif
|   |
|   \---MEOCS
|       |   allclasses-frame.html
|       |   allclasses-noframe.html
|       |   constant-values.html
|       |   deprecated-list.html
|       |   help-doc.html
|       |   index-all.html
|       |   index.html
|       |   overview-tree.html
```

```
|       |  package-list
|       |  serialized-form.html
|       |  stylesheet.css
|       |
|       +---resources
|       |       background.gif
|       |       tab.gif
|       |       titlebar.gif
|       |       titlebar_end.gif
|       |
|       \---start
|               CamReception.html
|               CommandSender.html
|               InUseMessage.html
|               MEOCS.html
|               MessageReceiver.html
|               RMIClient.html
|               TssInfo.html
|
\---WEB-INF
        glassfish-web.xml
        web.xml
```

F.4 Web version – Front End server

Prerequisites:

- Apache server installed

Procedures:

- The 2 following Virtual Hosts included in Apache configuration file “httpd-ssl.conf” :

Virtual Host 1

```
<VirtualHost _default_:port>
DocumentRoot "C:/xampp/htdocs"
ServerName IP_Address:port
ServerAdmin admin@IP_Address
ErrorLog "C:/xampp/apache/logs/error.log"
TransferLog "C:/xampp/apache/logs/access.log"
SSLEngine on
SSLProxyEngine on
SSLCertificateFile "conf/ssl.crt/server.crt"
SSLCertificateKeyFile "conf/ssl.key/server.key"
SSLCACertificatePath "c:/xampp/apache/conf/ssl.crt"

<FilesMatch "\.(cgi|shtml|phtml|php)$">
   SSLOptions +StdEnvVars
</FilesMatch>
<Directory "C:/xampp/apache/cgi-bin">
   SSLOptions +StdEnvVars
</Directory>

SSLOptions StrictRequire
SSLProtocol all -SSLv2
SSLProxyVerify none
SSLProxyCheckPeerCN off

   ServerName IP_Address:port
   ServerAlias www.WebsiteName.com

ProxyRequests off
<Proxy *>
Order deny,allow
Allow from all
</Proxy>
```

```
ProxyPass /InternetGasturbine https://IP_Address_of_Back_End:port/InternetGasturbine
ProxyPassReverse /InternetGasturbine  https://IP_Address_of_Back_End:port/InternetGasturbine
ProxyHTMLURLMap https://IP_Address_of_Back_End:port/InternetGasturbine

<Location /InternetGasturbine/>
    ProxyPassReverse /
    ProxyHTMLEnable On
    ProxyHTMLURLMap  /     /InternetGasturbine/
    RequestHeader    unset  Accept-Encoding
</Location>
</VirtualHost>
```

Virtual Host 2

```
<VirtualHost _default_:port>
DocumentRoot "C:/xampp/htdocs"
ServerName IP_Address:port
ServerAdmin admin@IP_Address
ErrorLog "C:/xampp/apache/logs/error.log"
TransferLog "C:/xampp/apache/logs/access.log"
SSLEngine on
SSLProxyEngine on
SSLCertificateFile "conf/ssl.crt/server.crt"
SSLCertificateKeyFile "conf/ssl.key/server.key"
SSLCACertificatePath "c:/xampp/apache/conf/ssl.crt"

<FilesMatch "\.(cgi|shtml|phtml|php)$">
   SSLOptions +StdEnvVars
</FilesMatch>
<Directory "C:/xampp/apache/cgi-bin">
   SSLOptions +StdEnvVars
</Directory>

SSLOptions StrictRequire
SSLProtocol all -SSLv2
SSLProxyVerify none
SSLProxyCheckPeerCN off

   ServerName IP_Address:port
   ServerAlias www.WebsiteName.com

ProxyRequests off
<Proxy *>
Order deny,allow
Allow from all
</Proxy>

ProxyPass /Internet_Gas_Turbine_Pages https://IP_Address_of_Back_End:port/Internet_Gas_Turbine_Pages
ProxyPassReverse /Internet_Gas_Turbine_Pages   https://IP_Address_of_Back_End:port/Internet_Gas_Turbine_Pages
ProxyHTMLURLMap https://IP_Address_of_Back_End:port/Internet_Gas_Turbine_Pages

<Location /Internet_Gas_Turbine_Pages/>
    ProxyPassReverse /
    ProxyHTMLEnable On
    ProxyHTMLURLMap  /     /Internet_Gas_Turbine_Pages/
    RequestHeader    unset  Accept-Encoding
</Location>
          </VirtualHost>
```

- Start the Apache server